ArduinGo: Final Paper for ITM 492

Alexander Keating

Hidra Antar

Alexander Wang

Craig Finnern

Our team was given the task of develop an electronic prototype covering the elements learned in the Embedded Systems course. These elements were electricity, data collection, data transmission, and data presentation. To do this, we decided to create an electronic version of the board game Go.  According to usgo.org, Go is an “ancient board game which takes simple elements: line and circle, black and white, stone and wood, combines them with simple rules and generates subtleties which have enthralled players for millennia.” To put it simply, in the game Go, players try to surround more territory than their opponent on the board.

In order to make this game electronic, we got rid of the physical game board that Go originated with and instead used LED rings called a NeoPixel to present our data for the data presentation portion of the project. These NeoPixel rings are made of 16 LEDs and each of these LEDs can be programmed to show any color light for any period of time. In order to use these NeoPixel rings to play the game go, we decided to make one light seem to move around the ring and then have the player press a button to stop the light and keep that specific LED on. In order to program the NeoPixel ring and power it, we used a small programmable computer called an Arduino UNO. The Arduino UNO is a great platform for this project because it not only small and compact, but it also can be used with various Digital and analog devices in order to make our game work.

While we could have had two buttons connected to the same NeoPixel ring, we also needed to transmit data as part of the project requirements. Therefore, we decided to use two rings that would communicate to one another using radios called XBEEs. We decided to use these radios because they are small, use very little power, and could transmit over a pretty far distance. The most important to us was the low power usage and the size of the radio itself because we wanted there to be handheld devices. With that, since it was a goal to be handheld, we also wanted the games or controllers to be battery powered, so the power usage was a big key in us picking them.

The button presses were how we collected data from the user and then sent it to the other player to also be displayed on their NeoPixel ring. When the button was pressed, the light on the controller would stop and the LED would stay lit of as stated earlier. However, since there were buttons on both controllers, each button was associated with a specific color. So, when button one was pressed, it would make the specific NeoPixel ring LED red on both NeoPixel rings by sending and collecting data when each button was pressed. Likewise, when button two was pressed, it would make that specific NeoPixel ring LED blue on both NeoPixel rings. We also collected the data that was sent between the two devices and stored them in a data structure that we will talk about later in the project.

While doing this project we ran into a lot of problems, such as Bounce, storing the LEDS in a data structure that was quick and easy to access, finding a way to relay information between the boards, making an efficient way to time the games, and making each board aware of which LED was lit on the opposing board. These were the issues that we spent the longest time trying to fix, there were a few others but these seemed to be the major issues that we found significant fixes for.

One major problem with using button presses as our entry method for the game is bounce. Bounce is the problem that occurs due to the fact that computers are much faster than any human being. So while a player might press a button for what seems like a very short time, Arduino UNO sees this press as the player holding the button down for an eternity. In order to work around this, when the player presses the button, the program loop is paused until the button is released so that only one LED is lit per button press. In the code we put a while loop, the while loop's condition was that while “red button” was “HIGH” or “pressed” to stay in that while loop. As soon as the button returned to its former state of being low, or not pressed, the program would break out of the while loop and return in the sequential order of whatever function was next. This works because the program is constantly looking for a button press so when it finally gets that button press it stores the data we need stored, whether it’s turning a light on or adding to an array, and then it waits until the user to release that button before continuing.

Now that bounce was fixed, we moved onto the next adventure. That was getting the LED’s stored in a data structure so that we could access that in the software to see what’s turned on and what’s turned off. So to make the LED spin in a circle we used a for loop, that would go for “i” amount of iterations and increment by one iteration each time, so it starts as i = 0, it goes through the loop once (all while checking for button presses) and then increments to i = 1 and so on until the loop reaches 15. Which at that point the counter is reset to 0 and the LED continues in the circle. It's important that you know the overarching view of the program so that we can compare items and there's an understanding of what we are speaking about. The first time we tried storing the number that related to which LED was lit we basically just stored whatever value that i was equal to, into the indices of that array. Later on we realized, that way was rather confusing and that in the most basic form an LED is either on, or it's off similar to binary, it's either a 1, or a 0. We decided that the best way to store the LED’s that were lit would be to just use 1 for on and 0 for off this way later down the line doing comparisons would be much more simple.

With Bounce under our belt, and the arrays being populated, it was time to start relaying the information between both boards. This was probably the largest challenge, for a while we couldn't even get the xBee to reset so that we could change their properties. Once we overcame that issue it was time to get down to business. We first had to make sure that we could turn lights on the opposing ring, which we actually did pretty quickly and that gave us a nice confidence boost. Once we tried to get both games running at the same time though and tried to make the red LED light up on the blue board and vice versa things got very tricky. We started to strip packets and tried accepting only the variable that we wanted to light up but that seemed nearly impossible as we were sending numerical values and every layer of the packet we would strip away just produced a completely different variable then what we were sending. We eventually needed to make some progress so we found a way to just accept the variable and nothing else, it wasn’t entirely accurate all the time but we needed to continue the project and find a way to work around it.

To make sure that each game only ran exactly 60 seconds, we made a global variable set equal to 2400. Arduino’s do things with milliseconds so everything had to be dropped down to that time. Every time the game would go through the loop it would drop that global variable by -10 milliseconds, this doesn't seem like a lot but it's timed out perfectly to 60 seconds. Then when that global variable is equal to 0, we break out of the loop, reset all variables to their former state, and give the user an option to start a new game by pressing the start button again.

Making both boards aware of what lights were supposed to be on and what variables needed to be equal was as simple as making sure that when a button was placed, the variable in that array was either reset to 0 or set to one, and the opposing array was set to either 0, or 1 depending on what button was being pressed. The same goes for the timer, we simply set out a signal when the timer variable was equal to 0 and it would stop all operations on the opposing board, tally up the arrays to determine winner, run the winners color in a fancy animation, and then reset all variables to a “newGame” state.

Overall, this project was a success, we managed to accomplish what we set off to do and made a playable, electronic, wireless version of the ancient Chinese game, Go. When the idea for the project first came up, our team didn’t think we would be running into very many problems, but we were in for a surprise. But through teamwork, and many long nights and early mornings working on our game, ArduinGo, we managed to finish it before the deadline caught up to us! This project was a great learning experience for each one of us as we had to teach ourselves and each other the knowledge needed to get ArduinGo working. Through making Bounce a thing of the past, learning to properly populate our arrays, relay new information quickly, set a 60 second timer, and make sure that each neo-pixel ring is displaying the correct information we really did make ArduinGo a success.