Light Battle

By: Matthew Woods and Benjamin Roberts

December 14, 2021

Final Product

Our group's final product was a success and we had to make no compromises to our original idea to finish our final product. We have a game that is focused on two computer-controlled players trying to connect their colored light on a light strip. At the end of the game will show how many points each player got and which player won. We did not need to display our game in console nor did our general concept of the game change in any shape or form. The details of the intricacies of how this project came together and how each aspect of this work will be explained thoroughly in its section of this research document. To talk about which specifics on who worked on which parts of the final product. Matthew Woods worked on constructing the game and making the AI that would handle the back-end data management. Ben Roberts worked on creating tools for using the light strip and front-end presentation of the project, as well as the assembly implementation.

Why this Project

This Project was the culmination of what everyone on the team is passionate about. Matthew, is really interested in AI. Matt previously worked on several personal projects that have AI for video games and this is why Matt was really into this idea. Ben also wanted to do Arduino project from the start. He already owned an Arduino Uno Wifi Rev. 2, and is constantly tinkering in his personal time. Both of us were able to lean into our strengths for this project.

Game and AI Challenges

- **Game Principles**

This game was meant to be a fun simple representation of the power of an Arduino being able to represent a simple game on an addressable light strip. That brings to question how simple should the game be and

how would it determined to keep track of the score. How would the AI players decide to interact with the data and how intelligent should it. If it was too intelligent then one player could win every time the game started, a topic that was constantly tested.

- ◇ **Debugging in Arduino**

The Arduino compiler is different than the ones Matt and Ben were used to using. Having around two hundred lines of code means that there is bound to be an error somewhere and when it came to debugging it was about a fifty percent chance that the Arduino compiler would point you to the line the error was occurring. Also, the Aurdino compiler did not have any complex debugging tools that Matt and Ben were used to like in Visual Studio's like watch list, breakpoints, etc.

- ◇ **Score Bug**

The most difficult bug during the production of the AI and game system was the scoring system. Scoring at first was very buggy. For example, player two would start off with a singular point for no discernible reason. When it came to checking the slots next to the player it would give improper outputs.

- ◇ **Hooking up the Arduino**

Once the AI was mostly complete, there were initial challenges in using the Arduino. Matthew had little experience using them, and originally believed the code was at fault, not the Arduino. Eventually Matthew consulted the group, and Ben was able to aid Matthew in allowing the Arduino to interface with his PC.

Game and AI Solutions

- ◇ **Constructing the Game**

Knowing the general idea of the game before starting to create it was helpful but still left Matt with some wiggle room with creating how it would operate. The biggest problem was how smart should the player be. Matt decided to have the decisions the players make still have a chance to be random so the game is not too predictable. Whether a player is winning or losing determines if they will be moving offensively or defensively. If a player is losing then they will might move offensively which means they will choose a spot next to one that they have already put on the board. If a player is winning they might play defensively which has them choose a spot next to one their opponent has placed on the board. In the method smartMove a small local array is used. This method loops through the gameData array to see if there are any possible

spots that meet the criteria of being offensive or defensive. if that spot exists it will be put in into the smaller array and after looping through the function will return a random value from the smaller array. if the value returned by smartMove is negative one, which means it was not set during the loop, then playerSelect will change it to a random spot. During the beginning, the AI's moves will most likely be random. While the game progresses, players will make more tactical decision as there are more places to gain or block points. The way the points system works is that every time a play chooses a spot that is next to three or more of its own color then gives that player a point. This scoring system fit the kind of game we were trying to make and therefore needed no need to make anymore complex than it already was. The way this was done was just by passing the spot chosen with a check method. This method would use while loops to check the left and right of the chosen spot. Overall this simple game served its purpose as being the back end of this intricate light show.

◇ **Using print and adapting to debug**

Primarily all of the debugging was done with Serial.print() when it came to the overall problems with the code at first. For the score bug, the system would print the variables used. With the variable, arrows would be printed to show the current state of the algorithm. It was found that for both checking the left and right, it was changing the value of my index by two. This error caused it to score points improperly and had some strange outcomes which baffled the programmer at first. The debugging for just this one problem took approximately an hour to debug.

The playerSelect method was also throwing errors. playerSelect works by using smartMove to output a possible move depending on whether the player is a currently losing or winning the game. smartMove also has the chance of outputting a value of negative one which tells playerSelect to change that value to a random number inside the game data array. There was a long time while Matt was debugging where it would randomly just run into an error and stop my code. The issue stemmed from the loop where a variable was needlessly being re-initialized because of another variable existing within the same scope. This problem was just solved with a careful reading through the algorithm, and it was determined to be dependent on randomization, since the error could change between runs.

◇ **Leveraging the Group**

Matthew Woods, was not familiar with using the Arduino Uno. Because of this issue, there was an initial roadblock to debugging the AI. Fortunately, the other group member, Ben Roberts was able to demonstrate how to connect the Arduino, while also showing the differences between the Arduino Esplora and Uno. To

expedite development, we used GitHub in combination with GitHub desktop, allowing for simple version control and parallel development. Matthew previously had experience with Git Bash, and the concepts were able to be translated easily, allowing for branches to be made for the major portions of code (AI, lighting control, and assembly).

◇ **Using smartMove efficiently**

Since we are using an Arduino Uno efficacy was in the back of both of our minds. When it came to the function with the most computation being smartMove it is important to get the most out of it as possible. The original idea of using AI was to have one function for offensive and defensive, like an offMove and defMove. But what Matt choose to do was have smartMove take care of most of the AI for both players. Since moving offensively and defensively in this game is really the same process but just depends if you checking for your color or the opponents, this use of smartMove was possible. So instead of calling two different functions, Depending on the game state smartMove will just be passed a different parameter and will handle all of the AI for the game.

Lighting

◇ **Protocols and Libraries**

When investigating how to control the light strip via an Arduino Uno, it very quickly became apparent that while control with assembly was possible, it was very discouraged. To replace the functionality, two libraries seemed to be most fitting: FastLED, and NeoPixel. The LED strip was configured as GRB over the WS2812B specification. The spec called for the use of just three wires: positive (connected via 5V header), ground (connected via ground header), and data (connected via digital PWM header). This is one of the most common methods for connecting LEDs because of its simplicity, thus FastLED and NeoPixel are both compatible. Ultimately NeoPixel was chosen as the library of choice because of its seemingly superior documentation, while also having quality of life functions.

◇ **Wiring Problems**

Currently two Arduinos were in play: a supplied Arduino Uno which Matt would use, and an Arduino Uno Wifi Rev. 2 that Ben already had. Ben held onto the light strip, and he noticed that all would be well, and then power would seemingly cut at random. As it turns out, the Arduino Uno Wifi Rev. 2 board has a minor power delivery defect, potentially caused by poor soldering. Similar to a smartphone cable that has

been used too long, two points of the board could move far enough away that electricity could not flow. This issue was compounded by the Arduino relying on its 5V USB-B port instead of the 12V barrel connector to power itself and the LEDs (though both ports did experience the intermittent issues as discovered later on).

To remedy the problem, the wifi board was eventually switched out in favor of a vanilla Arduino Uno board. This board also had the benefit of documentation, and a blink function was added. This function would allow for a visual indicator for when one of the AI scored a point via the on-board LED. This LED is low power, and soldered directly to the board, providing fewer points of potential failure. Initially this was written in C style syntax, though was converted to assembly since it also worked as a feature for the user. There was a bug regarding scoring, and scoring could also be confusing to some. So the LED allows for easy score keeping by the user, while also provided simple debugging for the programmers.

◇ **Software Development and Design**

Once wiring was solved, features had to be finalized. The group settled on an LED simply turning on at the specified power when a move was made, and only the ending sequence would have animations. Doing single moves was made easy by NeoPixel since the library knew the state of each pixel and could be gotten with a single function call. Unfortunately, NeoPixel stores the values in a packed integer, with the documentation specifically stating that it was impossibly to extract the exact GRB values from it, preventing the AI from knowing which LED had which player without having a secondary array to keep track for itself.

When it came to the ending sequence, it had to be very obvious that the game ended, especially since if the user overcame voltage limitations, could be hundreds of LEDs long. To allow for expansion while making it obvious the game ended, a dramatic change to the appearance of the LEDs would need to take place. It starts by turning off the LEDs, and simultaneously revealing AI 1's score from LED 0, and AI 2's score from the last possible LED. The LEDs are revealed two at a time (one for each player) until the number of LEDs lit corresponds to the score for each player (designated by color). The winner's LEDs the blink five times to indicate a winner. However it is possible that the two AI's will tie, which causes all the LEDs to blink with the combined color. Currently the two AI are red and green, so the resulting color is yellow.

◇ **LED Bugs**

Potentially the worst of the bugs were leftover code from prior debugging. One such bug was from testing syntax, leaving a single purple LED on during the win sequence. This issue is what inspired the move to the Arduino Uno board, since in trying to determine the source, the LED turned off. Thinking it was a completely separate section of code, it was debugged and commented line-by-line. In reality, the Arduino

had simply lost power, when that section of code was reached, and the real issue was an old line of code remaining at the start of the sequence.

Assembly

◇ **Documentation**

The biggest problem while attempting to insert assembly into the codebase was easy to understand documentation. While the group had some experience in three flavors of x86 assembly, the transition to AVR was difficult. Ben Roberts was tasked with scavenging for documentation, and what was found was too dense or irrelevant to be implemented quickly. Though what Ben did find were examples. From Ben's testing, he found that the same assembly would run on the Arduino Uno, but not the Arduino Uno Wifi Rev. 2. This discrepancy showed a difference in the processor, and forced any implementation of assembly to be held off as long as possible. When it was time to do so, the Arduino Uno was selected because of its wider adoption. The book "Assembly Programming and Computer Architecture for Software Engineers" textbook proved to be useful because of its example of AVR assembly.

◇ **Alternative Implementations**

Originally the goal was to implement other functions with assembly, specifically AISetup, togglePlayer, and getEnemyNumber, however significant issues were encountered. The most potent of which was the difficulty of passing variables defined within a function to the code written in assembly. Global variables worked out-of-the-box, but there were few of these. A workaround was considered, having a storing global variable, reference that in assembly, and then update the original variable with the stored one. Initial implementations were far from elegant, and it was eventually scrapped for C style. It ended up not being a problem since there was plenty of memory available, and the Arduino Uno had no problem with performance throughout the development of the project.

Light Battle in Action: `https://youtu.be/ufeQ4xU_ToU`