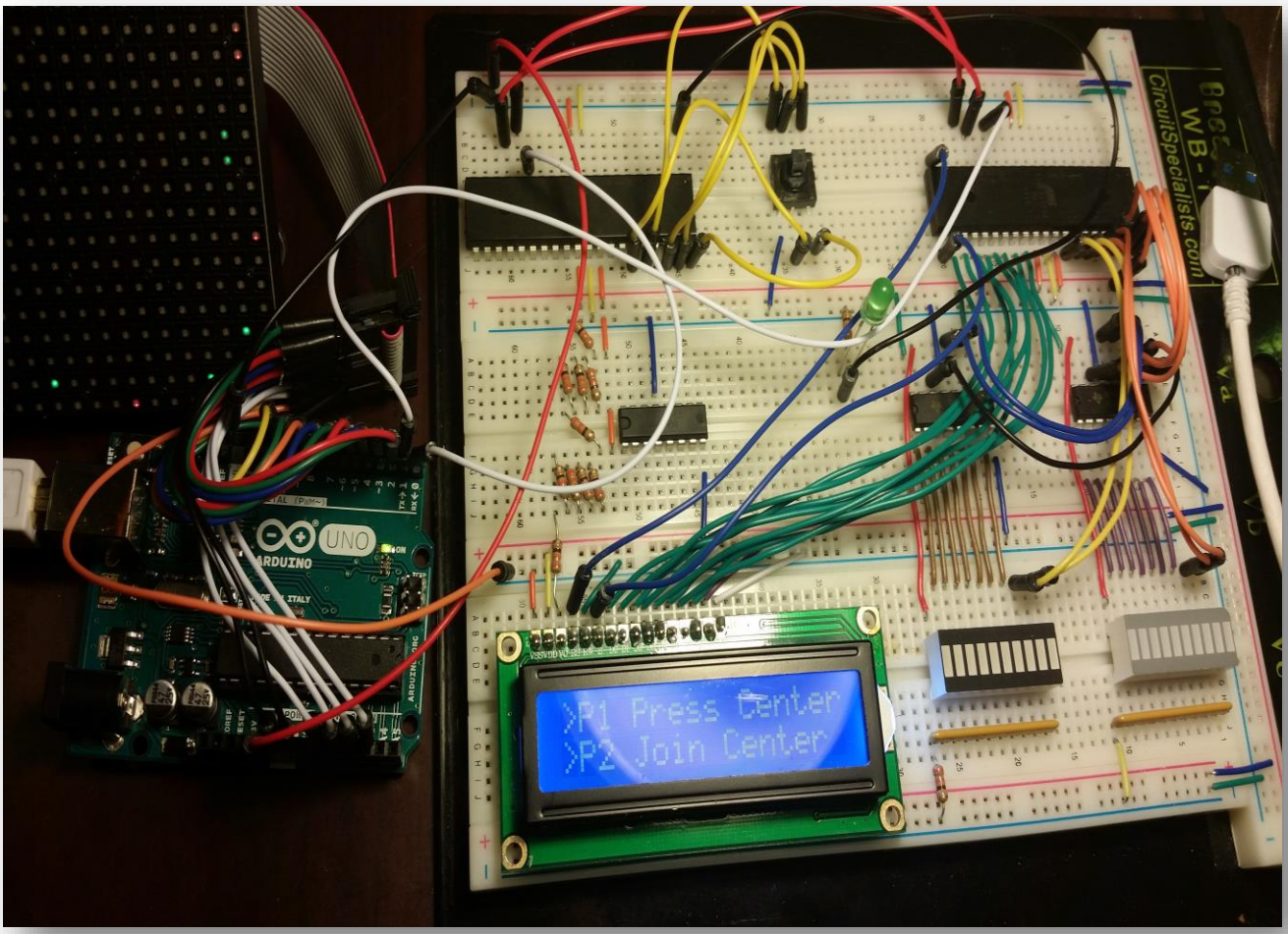
Wiring Setup



Wiring Explanation

- **ATMega1284 – UI System**

  The wiring is set up such that the Microcontroller on the right is connected to the UI system. It sends uses USART to receive event signals from the Arduino, which controls the game logic.

  PORT.A: Dedicated to controlling the Player Health Bar. Originally intended for two players, the UI needed two LED bars to light up totaling 20 outputs. The port has A[0:3] connected to a shift register. This shift register is daisy-chained to another shift register, totaling to 16 outputs. With four outputs needed with four leftover pins on PORTA, I connected the last four pinouts with the remaining LEDs.

  In the code, I transmitted the first 8 life bars from both players to their respective interfaces. This was done by using the left eight bits as P2's life output (after being converted from its value), and the right eight bits as P1's output. The last two bars of life were output separately from the shift registers.

  PORT.B: Used for debugging. Potentially could have been player input.

  PORT.C: Is used to control the LCD Display. It's traditionally wired as per the specs in Lab 5.

  PORT.D: Dedicated to USART transmissions. Only the RX was wired, since it did not require to send out data.

- **ATMega1284 – Input System**

  The left ATMega1284 connects to the player input controller. This was needed because the Arduino nor the UI ATMega contained enough pins to allow input. With one player, it was possible to integrate Player input along with the UI Microcontroller. With two players, as this game was originally designed, there were not enough input pins to be used on a single AtMega1284.

  PORT.A: Reads in Player 1 Input. Stores the signal into a variable (each bit represents a button being pressed), and sends it out to the Arduino via USART RX0.

  PORT.B: Used for debugging.

  PORT.C: Unused. Originally, it would have read in Player 2 Input. Stores the signal into a variable (each bit represents a button being pressed), and sends it out to the Arduino via USART RX1.

  PORT.D: Dedicated to USART transmissions. Only TX0 is used. If 2 Players were used, then TX1 would also have been wired to the Arduino.

- **Arduino UNO – Game Logic**

  The Arduino UNO controls both the Game Logic and the LED Matrix. The 32x32 LED Matrix consumes a massive amount of SRAM, so many design choices had to be made. Further details in the "Design Choices" section. It uses the ATMega328P.

  Digital Pins [9:2]: Used control the LED Matrix. Functionality of it is abstracted into a library published by Adafruit. According to the documentation, it contains six pins, two for each color for RGB control. The other three are used for Output Enabling, a Latch to signal the end of a row of data, and a Clock input.

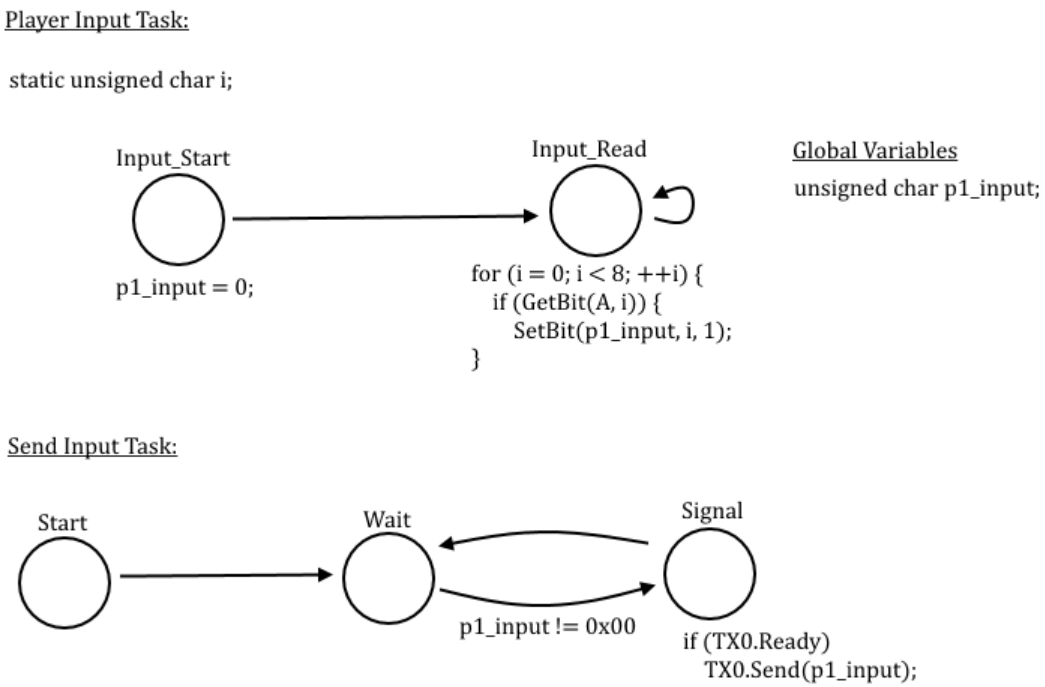
  Digital Pins [1:0]: The only two hardware pins on the UNO that control Serial transmissions. As such, Pin 1 functions as TX to the UI ATMega1284, while Pin 0 functions as RX to the UI ATMega1284.

  Analog Pins [0:3]: Also used to control the LED Matrix. According to the documentation, these control the data being displayed onto the matrix.

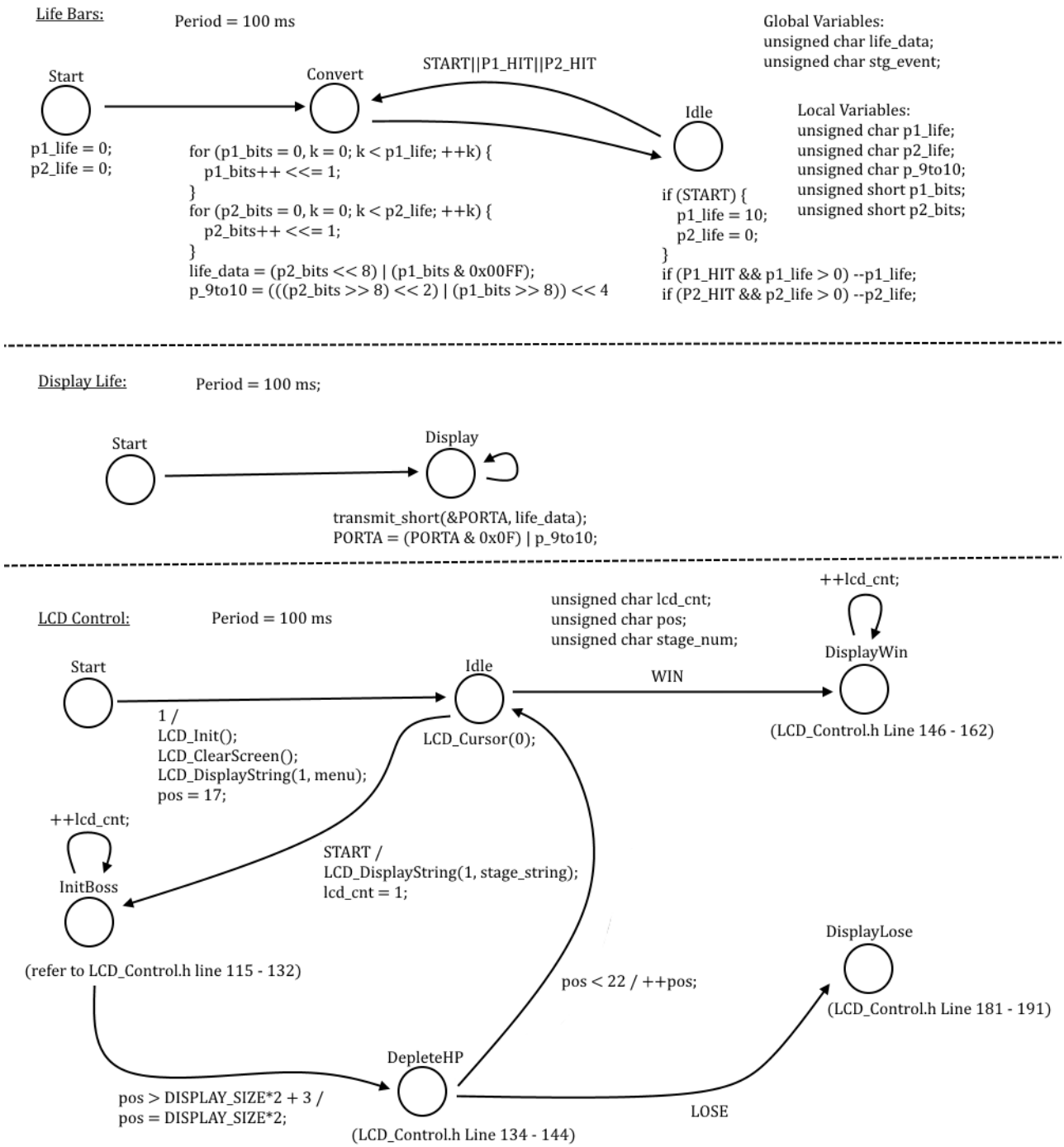
  GND/VCC: These pins are wired to both the LED Matrix and the Breadboard. The LED Matrix required an equal voltage be supplied to all three GND pins, otherwise graphical glitches would occur. Due to this, I was unable to use the YewRobot that IEEE supplied due to the unequal power supply. In order to simultaneously power the Arduino UNO and the two ATMega1284s, it was necessary to power breadboard with the energy routed into the Arduino from USB power by attaching the 5V and GND to the breadboard.

State Machine Drawings:

- **Player Input**

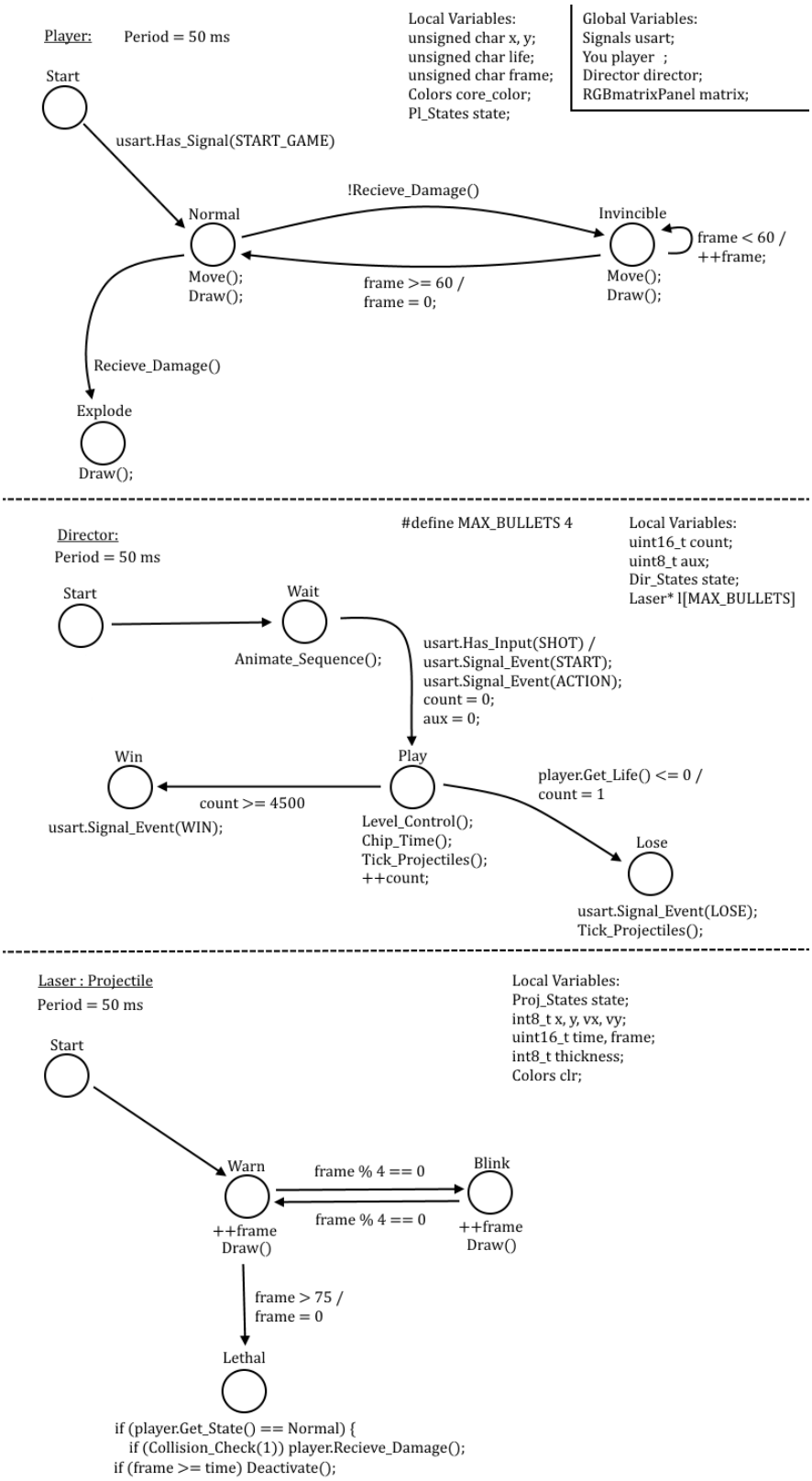


- **User Interface**



- **Display Logic**

  *Written in C++ using the same template, except integrating each state machine into a class. As such, the code will use member functions in each state action. Please refer to each class's header file for the functions in each state. This is to save space.



Known Bugs and Errors:

Regarding the known bugs and issues with the project, there are a number of mention:

- The Arduino UNO can only hold so much SRAM at once – specifically 2k. The biggest problem dealt with is the RGB matrix taking up around 1600 bytes of RAM, leaving me to program the game logic using the 400 bytes of RAM left to me. Because of this limitation in RAM, I had to optimize my variables and code to a huge extent. Even then, I did was not able to completely fit everything in under 400 bytes. The best I could have done was a little bit over. Because of this, the RGB matrix will have graphical glitches while playing the game. It does not interfere with the actual game functionality, so I considered it acceptable.
- USART does not do a very good job at keeping byte accuracy. As a result, sometimes signals sent from one microcontroller to the next will have some its bits dropped. This leads to shifted signals, which with my implementation, will lead to the wrong response from the receiver microcontroller.
- Because of the above, the life bars might not initialize correctly, and they will not decrement with the signal sent from the Arduino.
- If you don't hold down the start button long enough, the UI may not initialize.

Video Demonstration:

YouTube Link: https://youtu.be/qoXb1yVT5YM *(turn on annotations and captions, please)*

Library Sources:

- Adafruit RGB-Matrix-Panel Library: https://github.com/adafruit/RGB-matrix-Panel
- Adafruit GFX-Library: https://github.com/adafruit/Adafruit-GFX-Library
- GetFreeRam() debugging function: from 'usamacpp' in StackOverflow
- Arduino Base Library: https://github.com/arduino/Arduino
- MsTimer2 Library: https://github.com/PaulStoffregen/MsTimer2
- EE120B ATMega1284 Libraries
- Transmit_data(), usart_ATMega1284.h: TA Josh Yuen

Written Instructions:

1. The project is powered directly from the Arduino. For power, plug it into a USB power source.
2. The design of the game is simple. Once turned on, press the joystick in the center of the board to start the game.
3. After the game has started, projectiles will begin to spawn. The goal of the game is to survive a certain amount of time. The UI will display the time left. The green LED Bar to the left will display the remainder of your health.
4. Movement is controlled by the joystick. Be careful – it might pop out of the breadboard and you'll probably get hit.
5. You have a maximum of 10 health. If you get hit 10 times, you will explode and it's game over.
6. Survive until the end of the counter and lasers will stop spawning. That signals your victory.