Prof. Dickerson

Tyler Wiseman

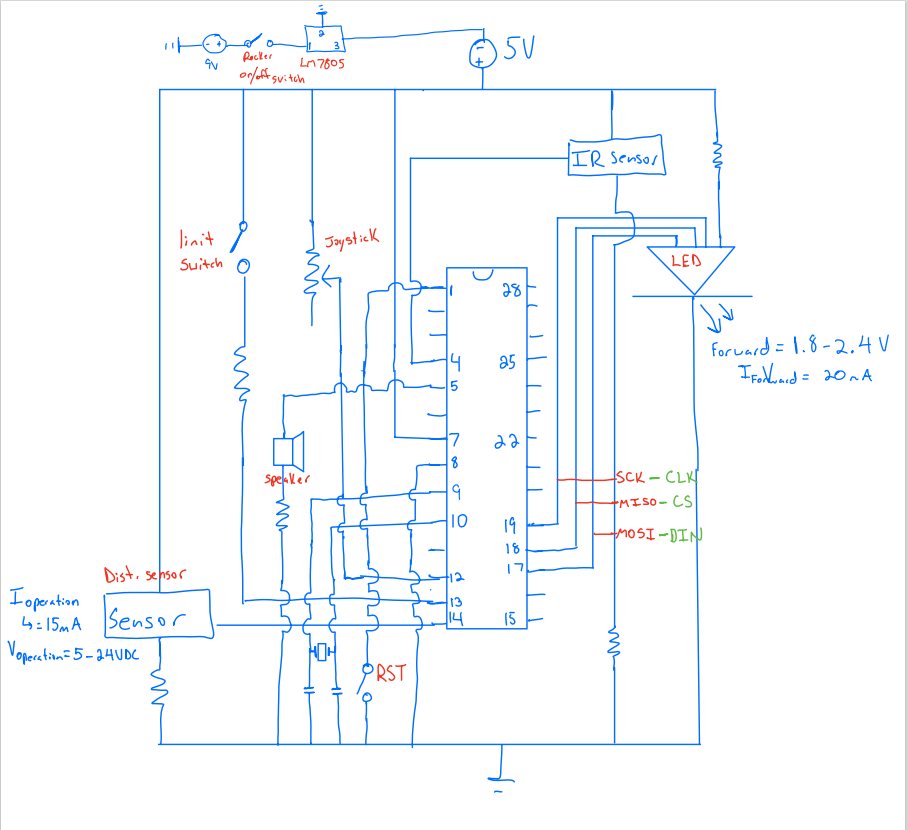
Keegan Borig

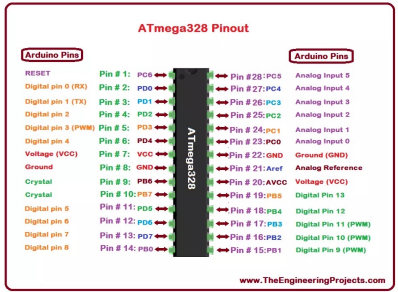
Daniel Stoller

ECE 1895

12/3/2023

**Punch-It!**





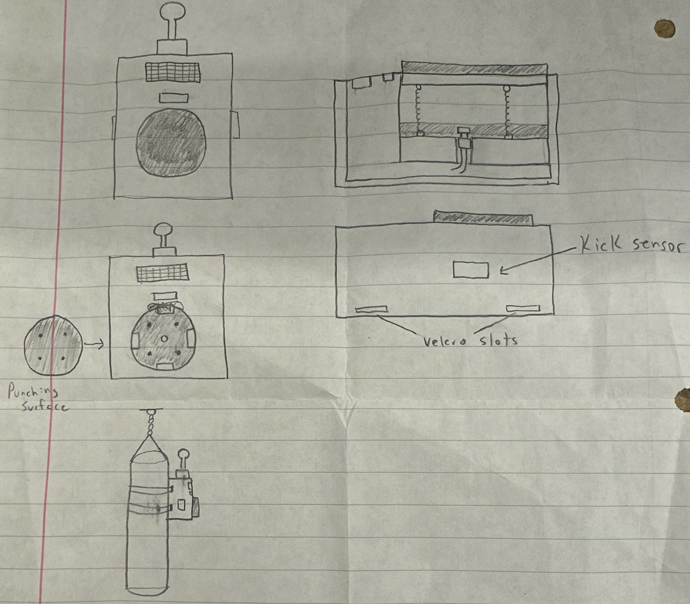
Our bop-it was designed to give more direction and functionality to punching bags. The goal was to make a fun workout that could teach key skills like reaction time and precision. We decided to create a bop-it that could be strapped to a punching bag and would have the 3 distinct actions of punch it, kick it, and dodge it. Punch It would direct the user to punch the foam area of the bop it, Kick It would involve the user kicking with their right leg, and Dodge It looks to see if the user is no longer in front of the bop it. This design will help people learn how to box, training their basic skills and their reactions speed.

To complete the Punch It, we created a box filled with foam and a limit switch inside. The limit switch will click when the foam is pressed in with enough force. Kick It is detected with an IR sensor that has a large radius of sight, allowing for a kick to be detected no matter the height of the kick. Finally, the Dodge It command utilizes a proximity sensor that can detect whether any object within a certain range is directly in front of it, allowing for the bop it to keep track of the user’s position.

Our plan had very little change throughout the design process. Small changes we made include switching from using a button to a limit switch for Kick It because it would be more likely to press since it is more sensitive. Additionally, our original plan was to use a joystick on the top of the bop it that when pulled down would start the game. This was to simulate “touching gloves” to start, just like in a boxing match. This did not end up working out, however, due to issues using the pins of the joystick and receiving the part very late into our process.

To accomplish this project, we utilized the ATMEGA pinout provided in assignment #8, as well as the data sheets provided to us for the more complicated components such as the two sensors and LED matrix display. These data sheets not only included the proper pinouts, but also example code and usage which made testing and getting started much easier.

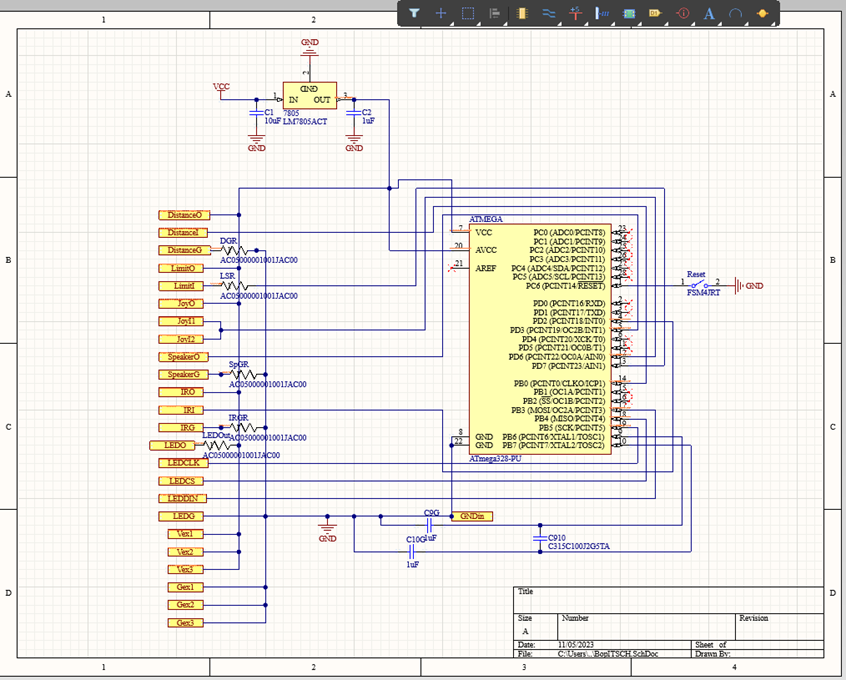
**Prototyping**

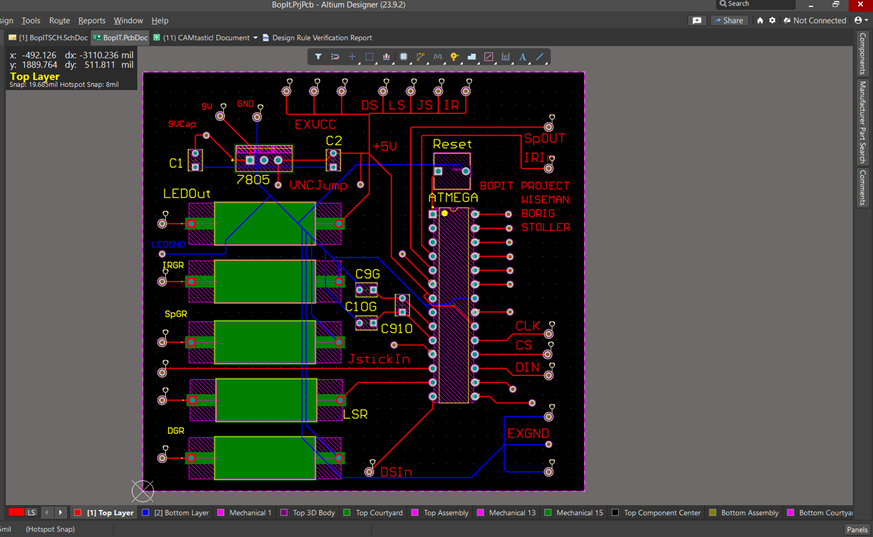


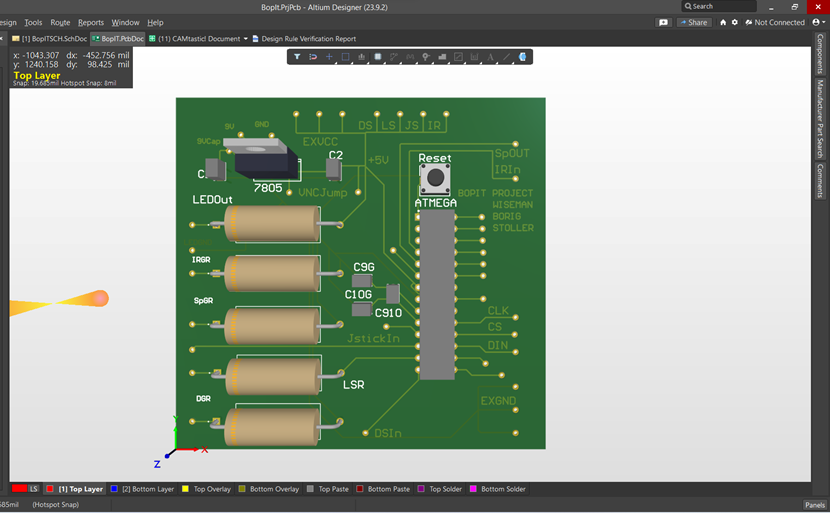
To prototype the enclosure, we started with a foamboard prototype. This proved to be useful because I could start over if we failed, and the material was very cheap. This prototype helped us find measurements and have a model to base our final enclosure off of. More specifically, we were able to find good sizes for the height, width, and punch area.

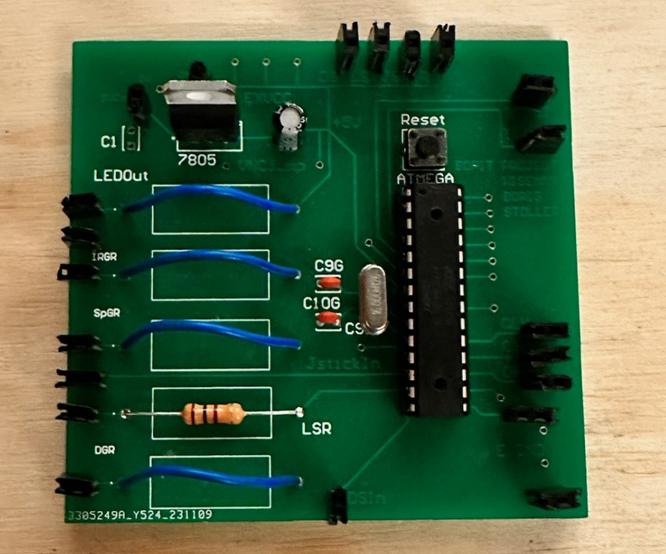
As for testing the electrical components, parts that did not require programming to function were easily tested using a breadboard and power supply. The limit switch, voltage regulation, on/off switch, and speakers all could be simply “plugged” in and their outputs measured using probes ensuring functionality. For those that required programming to function, once the code had been written, the ATMEGA was programmed via uploading test code from the Adafruit usbtiny boards directly connected to our laptops. Once the chip had been programmed, we would then also test each component on breadboard using the voltage regulated 9V battery and a programmed ATMEGA.

**PCB Design**









The design of the PCB kept in mind our current testing phase. As parts were still awaiting arrival, the PCB had to be made such that there were extra 5V power ports as well as extra ground ports and access to additional ATMEGA pins in case of emergency pin swapping. This also explains the usage of jumper ports and resistors placed between the peripheral inputs and ground.

The PCB uses a 7805 voltage regulator with parallel capacitors to take power supplied from a 9V battery and bring its output to 5V to be used by all circuit components. The ATMEGA itself utilizes an H-Bridge with capacitors to regulate the internal clock of the ATMEGA. There is also a reset button placed between pin 1 and ground if reset is necessary.

The peripherals include 2 speakers, an 8x8 LED matrix display, a limit switch, a proximity sensor, and an infrared sensor. The 2 speakers are connected from a PWM output of the ATMEGA (digital pin 3) into ground with both speaker leads being soldered together to simplify code and save pin space. The LED matrix’s pinouts connect to the ATMEGA’S power, ground, and three MOSI/MISO/SCK pins (pins 17-19). The limit switch, utilized for the “punch” action, acts as a pulldown connecting digital pin 7 to ground. The proximity sensor, used for “dodge”, is connected from power to ground with an output pin connected to digital pin 8. The infrared sensor, chosen for the “kick” action, like the proximity sensor has a power, ground, and an output pin connected to digital pin 2.

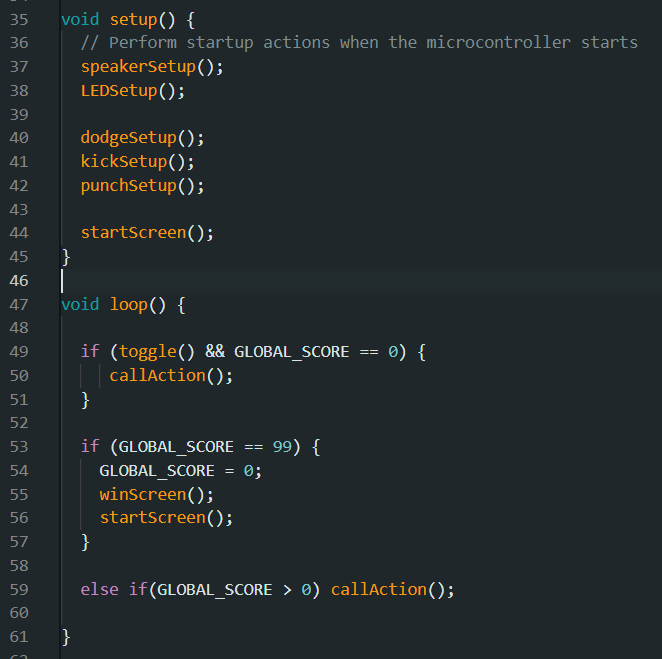
The overall design and testing process of this PCB went relatively smoothly. Most of the parts ordered were tested on breadboard first, once finished these parts were ported into the PCB with a programmed ATMEGA inserted to ensure the component functioned properly on the circuit board. The initial board/schematic design functioned entirely as intended.

**Software**

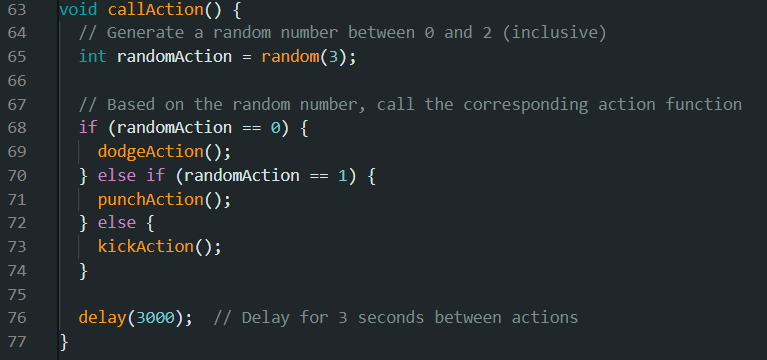
We employed the Arduino IDE to program our microcontroller. We include 2 libraries: one that helped with programming the speakers, and the other that made handling a limit switch easy. Next, we declared global variables that we can access and modify across various functions, like GLOBAL\_SCORE. It is also important to note the numbers used for GLOBAL\_ALLOWED\_TIME and GLOBAL\_TIME\_DECREMENT. For the first round of the game, users have 5 seconds to complete the action. With each successful round, the time allowed decreases by 25 ms. We additionally defined some of the pins here. Through testing, we found that some pins being declared at the beginning of the code helped issues that came up. From here, we declared our functions as shown below.



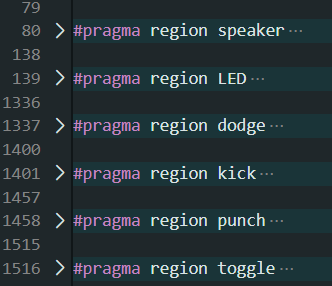
We then implemented the setup() and loop() functions which are default to Arduino. Setup() was a matter of calling other setup functions for the LED display, speakers, etc, and then loading the start screen. Loop() begins by checking if the game hasn’t begun yet. If it hasn’t it will continuously check for the start action, toggle(), which is simply the same thing as the punch action. Our loop function will also check to see if the player has won, and then display the corresponding win screen and reset the game. If neither of those conditions are true, loop() will choose the next action to complete with callAction(). See below for a reference to the setup() and loop() function.



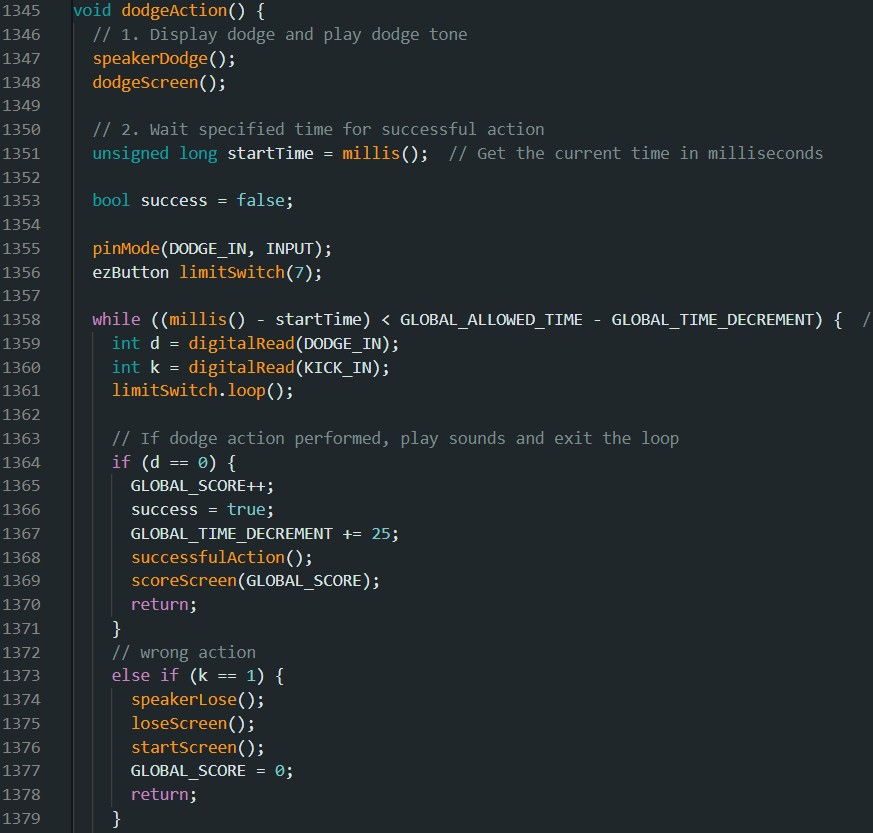
callAction() randomly chooses an action. If the action is completed in time, it will wait 3 seconds before calling the next action. See below for the implementation.



Following this overhead code are our implementations for each action, the speaker, the LED display, and the toggle. As shown by the line numbering, code readability quickly became important, so we organized this code by declaring regions.



Please reference the screenshot below while reading this explanation. I will only be explaining the code for one action, as explaining the code for all would become redundant. When dodgeAction() is called by callAction(), the speaker will play a distinct tune and the LED display will show the dodge animation. We start a clock, then redeclare pins in lines 1355 and 1356. Doing so simply fixed some bugs. A while loop begins which constantly checks if the allowed time has expired. We read in the dodge, kick, and punch values in. We first check to see if the user dodged. If they did, the success tune is played, the score screen reflects the gain of a point, and we return to callAction(). If a wrong action happens, the lose tune plays, the lose screen appears, the start screen then appears, and the score is set back to 0. The wrong action for punch (if user punches when prompted to dodge) is not shown but follows the same style. Similarly, the program calls all the losing functions and resets the game after the while loop exits, meaning the user did not complete the action in time.



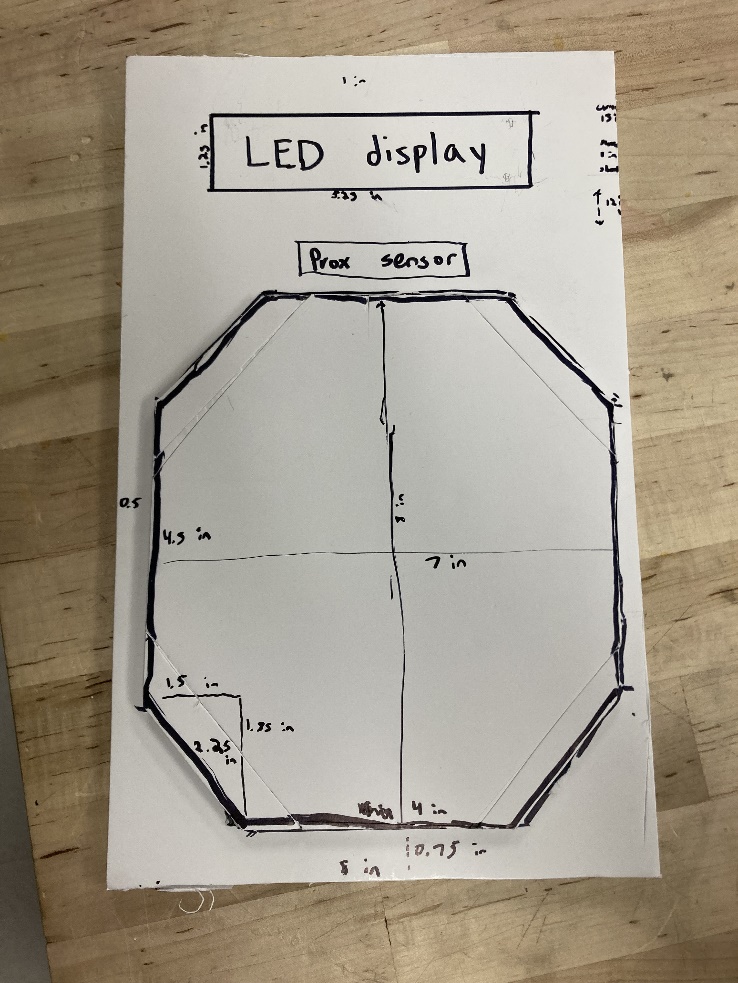
We used an 8x32 LED matrix to display our messages. We chose this because the user was going to be a few feet away from the enclosure, so we needed something big and bright to allow the user to see the messages easily. When creating the functions, we expected to be able to create animations, however this proved challenging. Changing every pixel on the board was very slow and it was very easy to see that every pixel was being reset. For example, we were going to have “Dodge” slide across the screen. This proved to be slow and ugly looking, however, as it was easy to see every pixel being reset and didn’t look smooth. To avoid this issue, we made the words stationary and small animations on the sides. Animations included punching gloves, an explosion, arrows, a person doing a flying kick. The only time we did a full screen animation was for the failing screen, but the slowness ended up working great for the glitch-type animation where it is supposed to look like something is wrong.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| You Lose | Glitching Animation |  |
|  |  |  |

These fun animations made things tricky at times. The microcontroller will not read any sensors if whatever display function has not been returned yet. There were several instances where the animation outlasted the allowed time, even if the player had been punching when prompted to punch. We had to play around and see what worked. We could not use the kick animation anymore, but we were able to keep some parts of the dodge and punch animations. We also adjusted the speed of how quickly the display disappeared. Obviously, we had to ensure functionality came before the aesthetics.

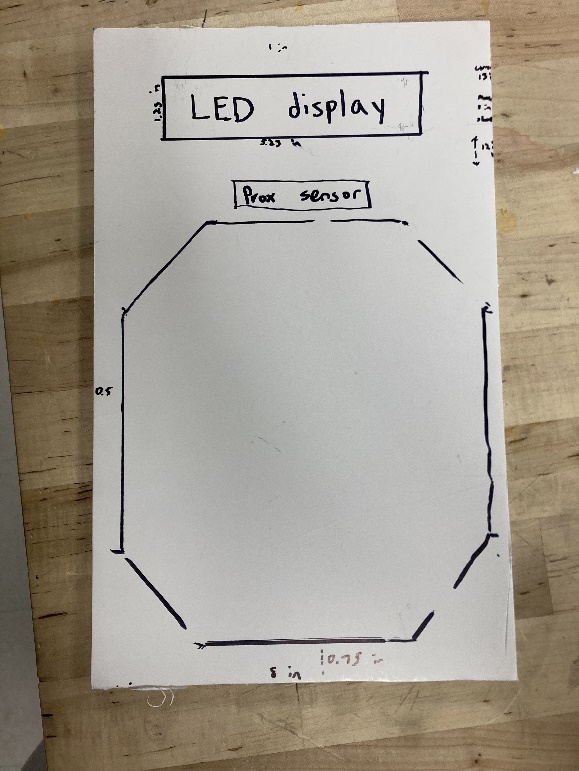
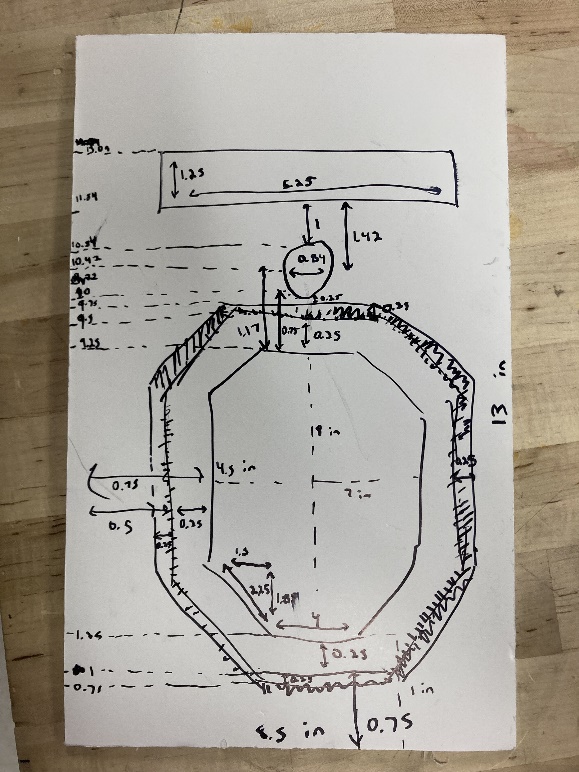
Our software was designed incrementally. We began with engineering the display. We tested that to ensure it worked and moved on. Then, we programmed the speaker and tested it. Next, we tested both together. We proceeded action by action, testing each module individually and then with the rest of the project. Although time consuming, this testing process was thorough, and we believe it saved us time overall, as we didn’t have miniscule yet crucial bugs to solve at the very end.

**Enclosure**

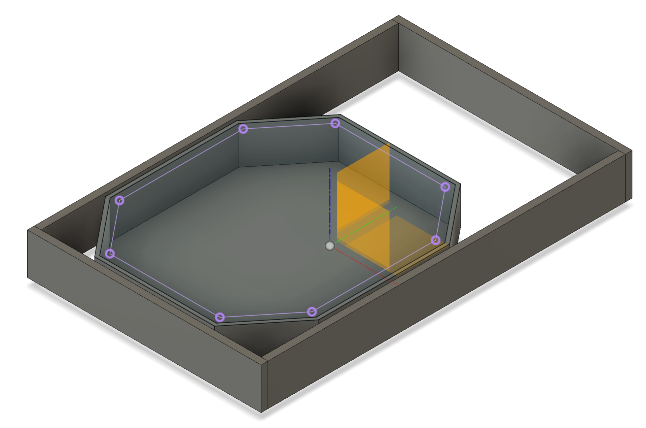


When designing the enclosure, we had to keep positioning of all the parts in mind. The punching area and dodge sensor needed to be at around shoulder height, the proximity sensor needed to be positioned in a way where the right leg would be the only object seen when going for a kick, and the LED display needed to be at eye level to be able to easily see the actions. With this in mind, we went with LED display at the top, then proximity sensor, then the punch area.

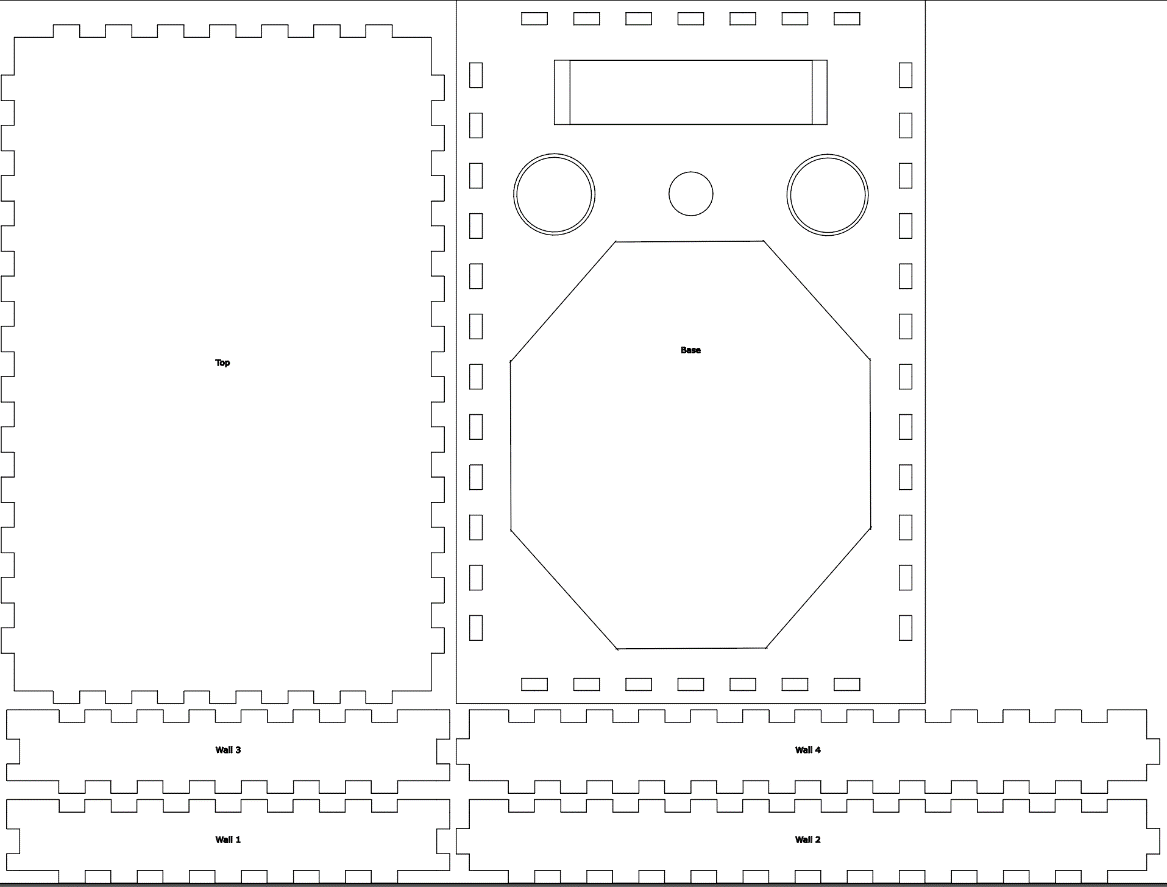
The other goal was to make a design that could withstand a punch. To fix this issue, we bought a roll of foam to absorb the impact of a punch. Also, we wanted to make sure the impact of the punch would be evenly distributed over a large area. This way, the impact wouldn’t cause a part to break. The final solution to this problem was putting a smaller box inside the enclosure, filling it up with foam, and placing a limit switch inside that would click if the punching area was pushed in with great force. We placed foam inside of the box to act as a cushion for your fist, a resistance for the push to allow only real punches to be detected, and a way for the punching area to bounce back out so the limit switch will be open again.



The enclosure was going to be big because of our large punch box, so laser cutting was the obvious choice since 3D printing would take forever and our parts were too unique to use a pre-existing enclosure. To create the prototype, we started with a foamboard cut out. This helped to figure out the total required length and height and to be able to physically test the punch area size to make sure it would be a big enough area without making it too large. Next, we 3D modeled the enclosure. This was used to figure out what the measurements would be in the z-direction (depth). While this didn’t end up being too useful for our group and probably wasn’t worth the time, it did help to communicate ideas with all the team members because we were able to point at the 3D model when expressing ideas and suggestions.



To make the box, we used Boxes.py for both the punching box and the enclosure. We were able to use the exact file for the punching box but had to add additional cuts to the enclosure including holes for the punching area, proximity sensor, speakers, and LED screen. Boxes.py was not reliable, however, as the walls were not scaled correctly. This required us to go into a separate application and correctly stretch the walls until the interlocking pieces fit correctly. We created an additional cut for the kick sensor. This cut was just 2 walls, a roof that left easy access to the potentiometers on the sensor, and a piece that the kick sensor could lean against, allowing it to lay in a downwards angle.



The final part about making the enclosure was ensuring that it could stay on a punching bag. To increase friction between the box and the bag, we put foam and non-slip pads on the back of the box. Additionally, we created slits on the back of the box to slip Velcro straps inside the box. These Velcro straps are used to wrap around the bag and create a tight connection.



**Assembly and System Integration**

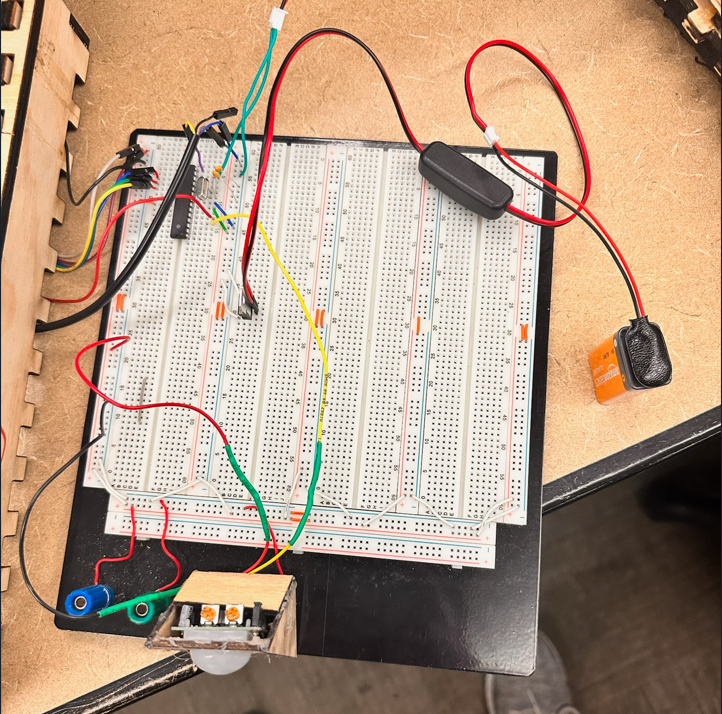


When constructing the final enclosure, we found it a good idea to go with a wooden and foam design as our project is designed to take a hit. The overall outer enclosure snaps together, thus a combination of wood glue and hot glue was used to hold the outer enclosure together. The punching area, which is where our limit switch resides, is screwed to the back piece of the enclosure and consists of foam surrounding the limit switch to both hold the switch in place and cushion the structure from impact. There is then a foam-covered piece of wood placed over the switch, which is padded to both reduce impact and protect the hand of the user. All foam was attached using the same combination of wood glue and hot glue.

To attach the device to a hanging heavy bag, holes were drilled into the back of the enclosure and zip ties were routed through. Four of these zip ties would make room to host two Velcro straps which are able to be wrapped around the punching bag. Also, on the back of the enclosure, more foam had been glued on to allow for more impact absorption as well as conform the rear of the enclosure to the bag more easily. Rubber nonslip pads were attached to this foam as well to prevent any sliding or slipping once attached to the bag.

The components of the Bop-It were all form fitted and/or hot glued to the top part of the enclosure with all wires hidden behind the LED/Speaker area. The PCB has its corners hot glued to the back piece of the enclosure to prevent any wires from falling out of the female pin headers on the board. Two complications arose from this wire-jamming. To remain hidden, some wires had to be lengthened to reach their destination pin. The other issue is that the cramming of wires caused some wires to become unplugged as the top of the enclosure smashed them down, and thus we had to strategically re-route wire paths to more snugly fit them into the appropriate pins.

**Design Testing**



As mentioned previously, the main testing plan once a part had arrived was to write test code for that specific part to ensure functionality. This code would then be uploaded to the ATMEGA and, on a breadboard, we would physically test inputs to see how each device reacted. After testing on breadboard with the programmer, we would then move to regulated 9V testing, and finally move to the PCB after testing continued to prove positive results.

We had several issues that came up with testing. One of those issues was with the IR sensor, which was responsible for the kick action. The IR sensor detects a wide field, while the proximity sensor’s detection is straight ahead, like a line. This caused many issues while testing. Sometimes we would get a failed attempt when trying to dodge, because our game thought we were trying kick. Our kick sensor was just sensitive, and we had to make sure nothing was close or moving near the kick sensor when moving. Luckily, our IR sensor had a potentiometer that changed the distance the IR sensor sensed. We were able to adjust this as needed.

Another issue was with our proximity sensor. Our proximity sensor showed a small red light when an object was in range. It worked independent of the other sensors, but not when it was connected to the entire system. No software changes would remedy this problem, as the red light was not coming on regardless of how close we were to it. Tyler came up with the idea of hooking it up to the oscilloscope and seeing how much power was running through it. It turned out that our 9V battery was only supplying ~4.5V to a 5V operating proximity sensor. The battery was replaced, and the case solved.

Debugging was made difficult by loose connections on our breadboard prototype. The ATMEGA was crowded by many wires, and wires would constantly come out. Repeatedly checking the breadboard for loose wires was difficult during the debugging process. It was hard to discriminate between a software problem and a hardware problem.

**Budget and Cost Analysis**

* Total Component Price: 131.09 x 10,000 = $1,310,900 (From BOM’S)
* Fabrication Cost Estimate: $6,682 (Utilized the link provided in assignment)
* Assembly/Labor Cost: $8,949 (Utilized the link provided in assignment)
* Enclosure Cost Estimate:
  + [Enclosure Area Est. (295.5 in)] x [Cost of 1/4in plywood ($0.0067/sqin)] = $1.97
  + x 10,000 = $19,700
* Total Cost (per 10k units): $1,346,231

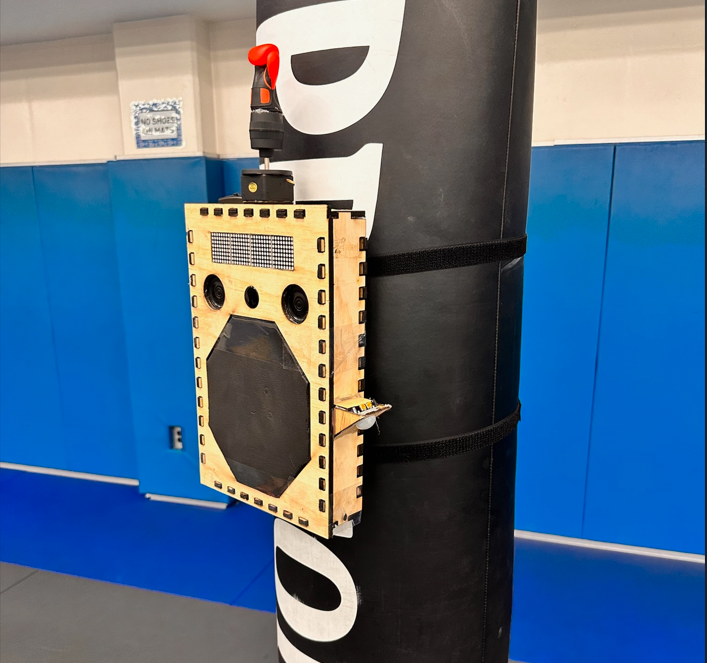
**Team Roles**

Keegan wrote all the code save for the LED matrix display functions. Daniel designed and built the enclosure and wrote the LED matrix display functions. In addition to helping design and build the enclosure as well, Tyler designed and assembled the PCB. To communicate, we used a text group chat and usually met after class. Towards the end of the build, we all worked together in the makerspace. We decided to split the tasks by category, having Tyler do all the hardware, Keegan do most of the software, and Daniel do the enclosure.

**Timeline**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Week 1 (Oct 16-22) | Week 2  (Oct 23-29) | Week 3  (Oct 30-Nov 4) | Week 4  (Nov 5-11) | Week 5  (Nov 12-18) | Week 6  (Nov 26-Dec 2) |
| -Parts List for BOM 1  -Finalizing design  -3D Modeling tutorial | -Testing parts  -Foamboard enclosure prototype  -BOM 2 | -PCB Design  -3D Model of enclosure | -Begin Laser Cutting. Make punching box  -Code speaker functions and begin coding actions | -LED display functions  -Complete all code and begin testing | -Finish Laser Cutting Enclosure. Make big enclosure and kick sensor box  -Finish software testing and program microcontroller |

**Summary/Conclusion**



Our Bop It was a success! All our actions worked, and incorrect actions worked as well. While Kick It is a bit temperamental because of the sensitivity of the sensor, in a perfect scenario it works very well. It is necessary to have someone holding the punching bag while playing, however, because the punching bag swings too much after being hit.

What we would change:

* Successfully use the joystick as the start action as originally intended
* Use a different IR (kick) sensor with a smaller range and radius
* Better integrate the LED functions with the interface software
* Create a physical signal when a punch is completed like a “click” or vibration so the user can know when their punch was hard enough and registered