Gesture Controlled Fighting Game

Team Members: Ray Dedhia, Petra-Juliahn Hernandez

GitHub: https://github.com/dihernandez/6.111

Overview

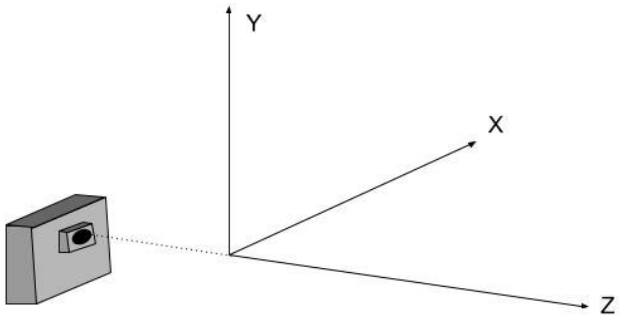
Our project is a player vs. player hand gesture controlled fighting game. The main components of this project are computer vision analysis to recognize the hand gestures, and the game logic/graphics. There are four hand gestures, each of which corresponds to one of the following actions: punching, kicking, and moving forward and moving backward. If the players are within contact distance, and one of the players makes a hand gesture corresponding to punching or kicking, points are deducted from the other player. The contact distance for hitting and kicking are different, and kicking does twice as much damage as hitting. Image analysis will be done with a camera that has an infrared filter from a floppy disc, an infrared LED attached to a glove, and a red LED attached to a glove.

Motivation

We chose this project because we believed it would be challenging and interesting to implement, but also feasible within our time constraints. Additionally, we thought a hand gesture controlled fighting would be an interactive and fun game to play.

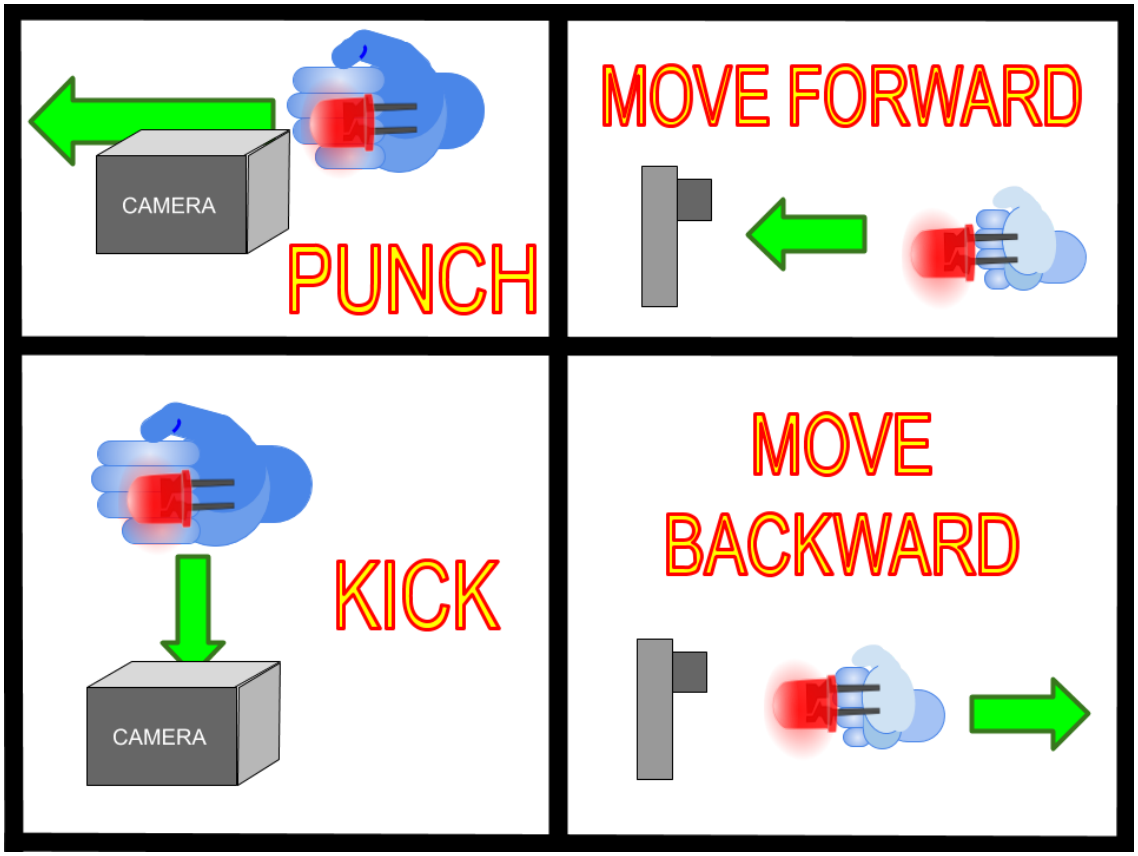
Implementation

Camera Vision (Ray)



Gestures

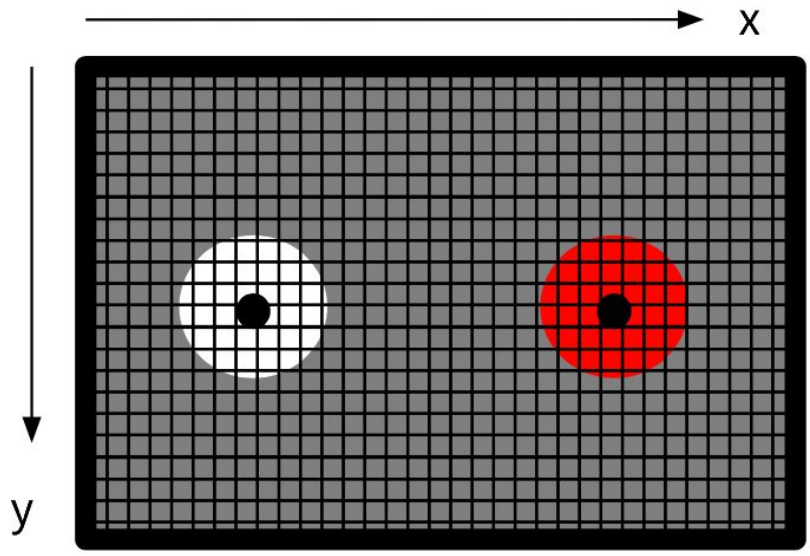
To punch, the player punches in front of the camera in the x direction (right or left). To kick, the player moves their hand in the y direction (up or down). To move forwards or backwards, the player moves their hand in the z direction (towards the camera or away from the camera). (Note: Forwards will correspond to the direction the sprite is facing (right or left), and backwards will correspond to the opposite direction.)



Motion Detection

Our setup consists of one camera with an infrared filter from a floppy disc attached to its front that is positioned facing the players. Both players sit in front of the camera with the hand that they are using to gesture motions (punching, kicking, moving forward, moving backward) in front of the camera. One player wears a glove with an infrared LED attached to it, and the other player wears a glove with a red LED attached to it.

The camera takes a video of the players' fighting hands. From each frame, for each LED, the number of pixels that meet the threshold for being from that LED is counted, as well as the sum of the x and y coordinates of those pixels. To extract the (x,y) location of the LEDs in the frame, the sum of the x and y coordinates are divided by the number of pixels detected from the LED using a divider IP module.



The rate of change of these coordinates over a set of 8 consecutive frames is used to calculate the average rate of change of the players' hands in the x and y directions, as well as the rate of change in the size of the players' LEDs. The magnitude and sign of the rates of changes are stored separately. This information is used to classify the hand gesture as one of the following states: none, punching, kicking, moving forwards, and moving backwards.

For instance, if the rate of change of change of an LED in the x direction is below the MAX_DX_KICK threshold, and the rate of change in the y direction is above the MIN_DY_KICK threshold, the action will be classified as a kick. Similarly, if the rate of change of the size of an LED is above the MAX_DSIZE threshold and is positive, that means the hand gesture is moving forward.

Parsing Video Data

For our project, we use a OV7670/OV7171 CAMERACHIP™. The OV7670/OV7171 CAMERACHIP™ is a low voltage image sensor with the functionality of a single-chip VGA camera and image processor, and can operate at up to 30 frames per second (FPS). We use the 12-bit color option. The camera module receives 8 bits at a time from the image sensor at some rate t_{PCLK}. One pixel takes two clock cycles to be received from the camera. Once the data for an

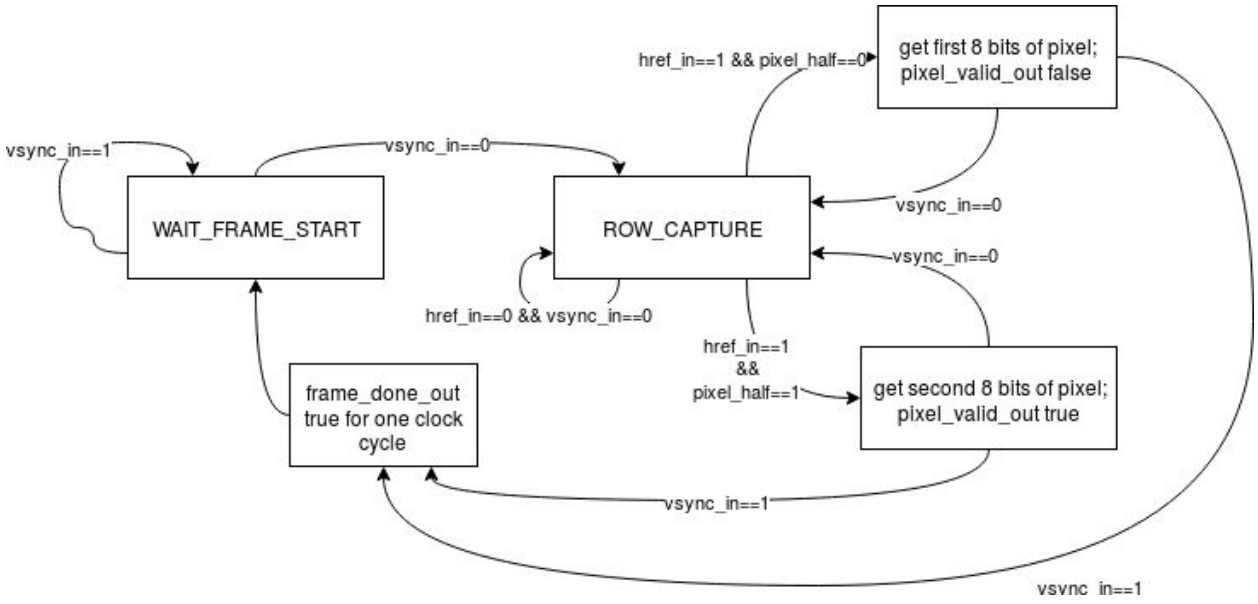
entire frame has been passed through and stored in memory, the camera_top module extracts the position and size data needed from the frame and saves it as variables.

The camera_top module also stores whether or not each LED was detected in the past 8 frames. After position data has been extracted from 8 frames, if the LED was detected in at least 5 of the past 8 frames, the camera_top module classifies the hand gestures from the video data. The module then outputs the updated states of the two sprites as the following boolean values: p1_punch, p1_kick, p1_moving_forward, p1_moving_backward, p2_punch, p2_kick, p2_moving_forward and p2_moving_backward.

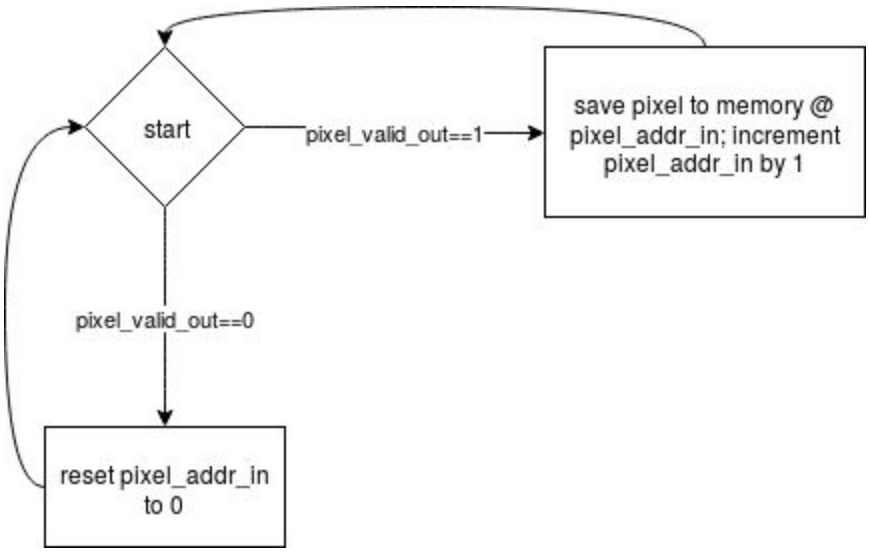
Clock Rates

We use two different clocks in this project: the video clock rate and the action clock rate. The video clock rate is the rate at which the FPGA receives frames of the video from the camera. The display clock rate (65 MHz) is the rate at which the game module receives updates regarding the state of the sprites, as well as the rate at which the display screen is updated.

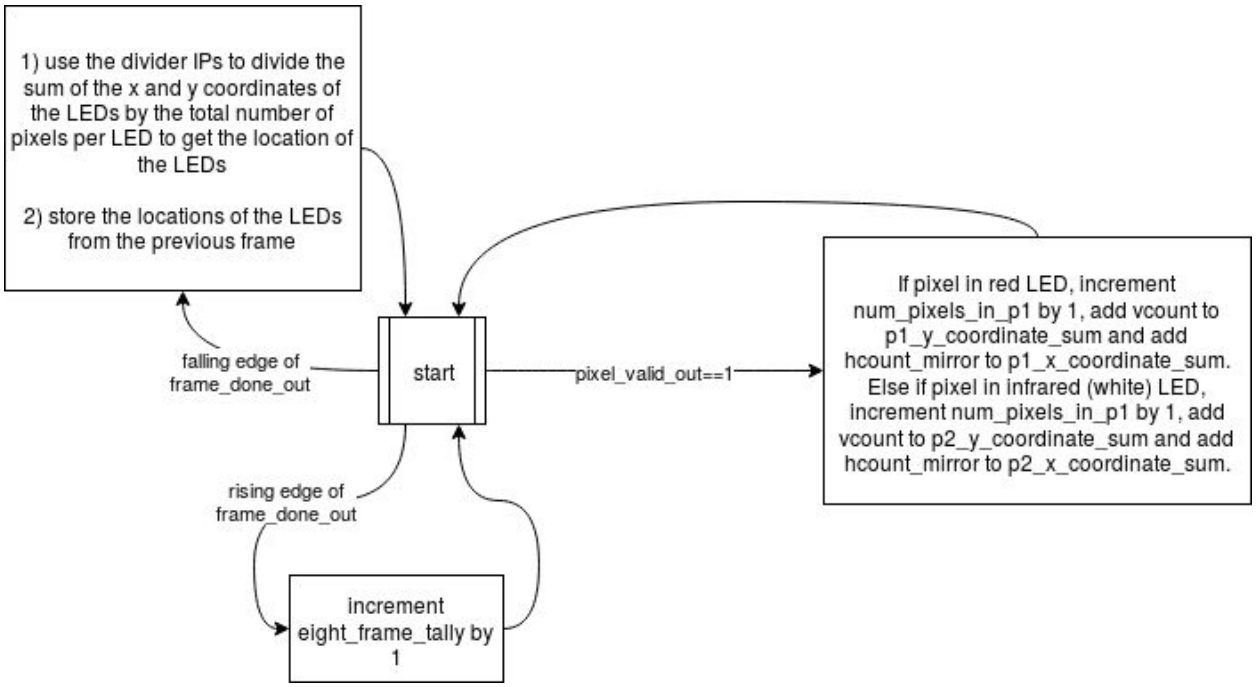
FSMs



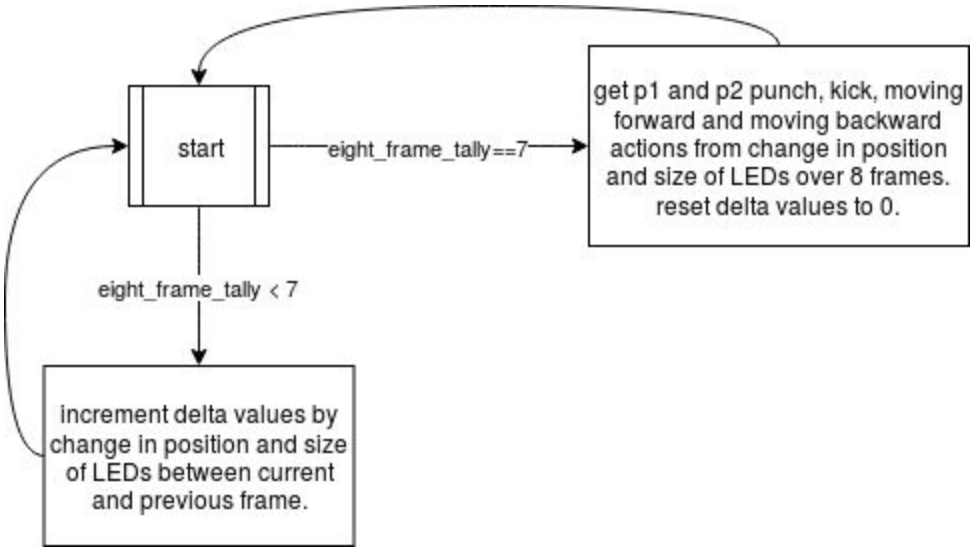
Camera FSM: Gets the pixels from the camera and outputs when the pixel output is valid and when the frame is done.



Memory FSM: When the pixel output of the camera module is valid, stores it in memory at the correct address.



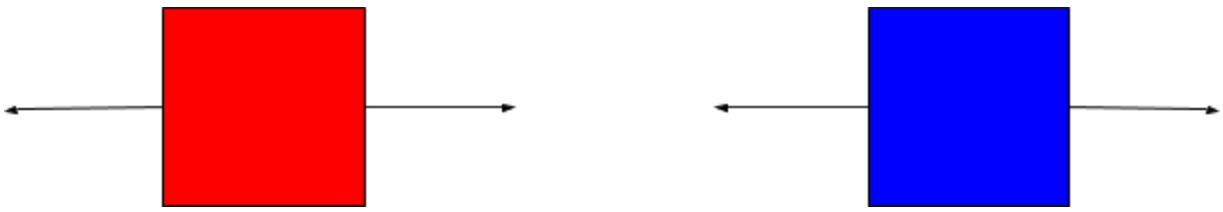
Extract location of LEDs FSM: When the pixel output of the camera module is valid, updates the number of pixels from each LED and the sum of their x and y coordinates. On the falling edge of frame_done_out uses those values to calculate the locations of the LEDs in the frame and store the previous locations of the LEDs in variables. Also increments eight_frame_tally on the rising edge of frame_done_out.



Get actions FSM: Calculates the change in location of the LEDs in the x and y direction and the change in size of the LEDs. On the eighth frame (when eight_frame_tally==7), uses those to determine which action the LED is making and set the boolean variables p1_punch, p1_kick, p1_moving_forward, p1_moving_backward, p2_punch, p2_kick, p2_moving_forward, and p2_moving_backward.

Gameplay (PJ)

Player Sprites and Movement module



We wanted to have small sprites on the screen to represent our players. To do this, we need to control the VGA output to display an image. The VGA gets input as a string of bits that control

the Red, Green, and Blue colours on the screen. The easiest image to render is a single pixel, but in lab 3, we were given a module for a simple rectangle which rendered the top left corner at the specified x-coordinate and y-coordinate.

These x- and y-coordinates are each controlled by the movement module, where the y value was some constant value that both of the sprites shared and the x value was determined by the previous value of x and the direction of motion indicated by the gesture of that player. For example, if the gesture was forward, the sprite would move forward 4 pixels. Backwards would give 4 pixels backward. We also used the btnr, btnl, btnd, and btnu buttons to control the sprites when the movement was not reasonably functional yet and the 16_r,g,b and 17_r,g,b lights to make sure that those buttons were registering presses. This was very helpful for debugging purposes.

Another input for this module was the hp module. If a player died, the original 64x64-pixel sprite would be replaced by a 64x16-pixel sprite, which would not be allowed to move. In that case, the winning player's sprite would still be able to move freely in a victory dance of sorts.

We also differentiated these sprites by giving them different colours: red (#F00) and blue (#00F).

Health Point Calculation

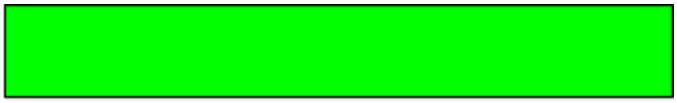
We also wanted to be able to see the effect that our punches and kicks were having on our game, so we needed to implement a health points module. The way we did this was that we set several parameters for initialization of our game. These were the starting HP of each player, the arm length, leg length, punch damage, and kick damage.

Players would only be able to damage their opponent if they were within range to do so. So if a player was further away than the length of their leg, then they should not be able to kick their opponent and thus should not be doing any damage to their opponent. Furthermore, the player's HP should not be able to go below zero, as this would cause underflow and would show the HP as a very large number instead of a small one, which would also make the logic for a health bar more difficult.

It was also noticed that from the camera module, any hit registered would stay registered for a total of 8 frames, which was a very large number, so I implemented an edge detector to prevent the game from killing players with one hit.

Before the health bar could be created, we needed to find some way to display the health points, so we decided to use the seven-segment display on the board in a similar way we used it in labs 2 and 4. It would show the value of the HP in hexadecimal on the display, which was sufficient for debugging.

Health Bars



As for the health bar generation, I modified the module for generating rectangles to allow the length and colour of the rectangle to change. The length of each health bar was 4x that player's HP, with an added 3 pixels at the end so that the health bar would not completely disappear when a player ran out of health.

The colour was represented by a list of red, green, and blue values. Each player would start out with a completely green health bar (#0F0) and on a punch, the red value would increase by 2, while the green value would decrease by that same amount, while a kick would result in a red increase of 4 and a green decrease of 4. I also checked the HP so that if a hit would result in a drop in HP low enough to kill a player that it would just change the colour of that health bar to pure red (#F00).

Challenges

Ray

I spent about two weeks trying to figure out the best way to detect player 1 and player 2's hands. At first, I was going to have the two players wear brightly colored gloves and use HSV thresholding to extract the locations of the players' hands as well as their size/distance from the camera. I implemented a module that converted the RGB output from the camera to HSV. However, Gim suggested using an infrared filter from a floppy disc and an infrared light, which resulted in an output that was much easier to track, so I ended up abandoning my rgb_to_hsv module.

Unfortunately, I faced a problem with this: all infrared lights show up as white through the floppy sic filter, and I needed to track two players. Fortunately, I realized that LEDs show up as red through the infrared filter, and that I could have one player use the infrared LED and the other player use the red LED. After this, I had the problem that when the red LED was close to the camera, it was detected as both red and white. After a few days of trying other options, I realized that if I put a tissue or piece of paper in front of the camera to diffuse the light, it would be detected as only red.

Another problem was figuring out what values to use for thresholding for the rate of change in the x and y directions to detect punches and kicks, the rate of change in the size of the LEDs to detect moving forwards or backwards, and for determining if a pixel was from the red LED or the infrared LED. I ended up using the switches to try out different values for the thresholds and pick the best ones.

Another problem was with using signed logic. I started out by trying to use signed logic to hold the values of the rates of change. However, since the outputs of my divider ips weren't signed, this messed up the signed logic and resulted in my values being incorrect. After a TA helped me figure out what was happening, I decided to track the magnitude and sign of my variables separately.

Lastly, I tried to implement an infinite impulse response (IIR) filter to smoothen my rate of change variables, but had difficulty figuring out how to deal with signs. I ended up trying to use the sign to convert my ranges of values from (-M, M) to (0, 2M), pass them through the IIR filter, and then convert them back. However, my code didn't work and I didn't have time to debug it.

PJ

When it came to my part of the project, there were many difficulties.

Firstly, we wanted many hardware things in our final design, but I was very unfamiliar with how to use many of them. I was unaware that the FPGA board was not capable of using alternating current, so finding cheap motors for potential haptic control was difficult. We also wanted to be able to control LEDs for feedback, but to control them, we would need to use bluetooth, a tool that I had no experience interfacing with, so I spent many hours researching bluetooth and coming up empty with regards to how to use it, so we never used it in the final design.

Secondly, there were many miscommunications between me and my partner. For almost a week, I was not aware that the gestures were represented for 8 frames as a high value, instead of only being represented for one frame. So having clear communication with your partner about how you represent your signals is a very important lesson.

One of the bugs that I came across was that my sprites would teleport when they were moving. This took several days to figure out, but it turned out to be a problem with the way that I had defined my characters to send out death signals and that I had incorrect bounds on the motion of my sprites. They would die every couple of motions and then not be able to move for a few cycles, but retain their health.

The above problem started happening again, but only when I changed the parameters of punch damage and kick damage. I sat with several LAs trying to figure out this bug to no success, but just leaving the value of the punches and kicks alone turned out to be pretty effective, so I didn't change them again and the problem went away.

Initially, our player sprites were supposed to be cute sprites we made ourselves, but figuring out how to convert our png files to the appropriate format to generate a COE file was not something that we should have been trying to figure out in the last week.

Lessons Learned & Advice

Ray

If I was doing this project again, I would plan ahead more with regards to getting parts. We ended up using the LEDs that 3.091 hands out as part of a homework assignment, and we ended up using tape and plastic forks to create our camera stand. In terms of advice I would give to future students taking this class, I would recommend starting early, and taking the time to plan everything out before you begin.

PJ

If I were to do this project again, my biggest piece of advice would be to encourage you to start as soon as possible on each piece of the project. You should be working on your part of the project and trying to improve your part even if you can't rely on the inputs from your partners' work. The FPGA has many other buttons and switches that you can use for inputs instead and these can be utilized to your advantage and for smoother introduction of other inputs. If your partner is using those buttons, you can use a switch to control whether the buttons/switches will be used for your debugging or your partners'. This was a very helpful realization I came to far too late in the process of the project.

My second most important piece of advice would be to communicate clearly with your partner about the format and timing of your signals would be. Clear diagrams may not be as clear as you think they are. A clock signal or a similar diagram for the signals you are sending with a clearly defined period can be a great time saver.

Furthermore, FSMs were a device that I did not use nearly enough in the final project. I realized with not enough time left that my design would have been greatly improved and much easier to debug had I put aside some extra time to use an FSM as initially intended.