**ECE 287 FINAL PROJECT**

**SNAKE GAME**

JAKE O’SHAUGHNESSY AND HAN LI

The creation of this game presented several problems we had to overcome. The biggest of which did not involve the implementation of the game itself. The biggest problem came with the VGA output, which proved more difficult to implement than we had imagined. In the end, we took some source code we found online, and implemented the component to work with the size and clock of our screen. Our screen was a 1920x1080 VGA screen with a 50/60 Hz, and following the chart they had online:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Resolution | Refresh  Rate | Pixel  Clock | Horizontal (pixel clocks) | | | | Vertical (pixel clocks) | | | |
| 1920x  1080 | 50/60  Hz | 193.16  MHz | Display | Front Porch | Sync  Pulse | Back  Porch | Display | Front  Porch | Sync  Pulse | Back  Porch |
| 1920 | 128 | 208 | 336 | 1200 | 1 | 3 | 38 |

Following that, however, the creation and movement of the square became fairly simple. A signal, direction, was created in order to keep track of where the square was headed. Following that, we implemented a clock and count to signal the movement of the square. We also found source code online to let us use the keyboard as an input. This was fairly easy to implement, however, we found several bugs with the keyboard such as when we hit the up arrow, the direction is turned to up for one movement, and then switched to down. We could only think to blame it on the bounce of the keyboard. Following this, we put a condition in which if the square hit the edge of the screen, it would reappear at the other side of the screen. Now that the head of the snake had been created, the next step was to create the food. It was very simple to create a different colored box, yet it proved more difficult to tell when the head of the snake moved over the food. Since the horizontal and vertical pixels are both divisible by 40 we made each square produced have a side length of 40 pixels so that the squares would be flush against the border if they reached it. Yet if we wanted to randomly generate a new position for the food if the snake head went over it, we wanted the position of the food to remain on a pixel divisible by 40. So we generated a number for the xpos between 0 and 1920/40, and the ypos between 0 and 1080/40, and then multiply those numbers by 40 to get an xpos and ypos that remains on that standard.

What we had yet to get to was developing multiple squares to represent the tail of the snake. We could not make one long rectangle because we needed to account for if the snake moved. The only way around that was to make the snake to be made up of many squares. However, we were unsure how to create a “length of snake” number of squares because we had to hardcode each square that we created. The rest of the rules of the game would have been easy to implement through arrays representing the x and y values each of the snakes body parts. Each square would in the body would take the position of the preceding square so that we would have a “follow the leader” effect that is always seen in this game. We would then be able to compare the position of the head with the positions of the rest of the body parts using these arrays, determining if the game would end. We had just gotten stuck on this problem and could not find a way around it.