**ATOMIC HAMSTERS - LESSON**

Nicole:

A major challenge this semester was getting an animation class working properly. For a lot of the semester, our Render section was very cluttered, and there was a lot of redundancy. As we added more and more features to the hamsters, such as increasing/decreasing size, increasing/decreasing speed, different coloured sprites based on user choice, etc., we decided that making an animation class was very important, because it was becoming very hard to cleanly handle the large amount of code in Render, and the lack of organization was making it hard to move around. Originally, we had created two classes, a hamster class and an aniSprite class. The hamster class handled the animation, and the aniSprite class took the direction instructions, hit detection, variable resetting, and more. This was working well, however a little redundant to keep passing the variables around. We ended up taking all the hamster class stuff into the aniSprite class, condensing it. This also condensed Render, because only an instance of aniSprite needs to be created now, not one of hamster as well. As of now, two aniSprites are created in each screen, and the coordinates and proper texture can be loaded. Then, aniMouse.move() is called, which updates the coordinates based on direction. Then, aniMouse.animation() is called. aniMouse.isHitS() checks hit detection, and if it is hit, aniMouse.outOfBounds() is called to move it back. The isHitS() also calls getThisRect(), which grabs the rectangle of the sprite and creates a new rectangle with corrected height and Y coordinates. This is returned and used for hit detection. Finally, once game is over, aniMouse.reset() is called to reset all variables. In the future, if I ever code another game, I will certainly make an animation class as one of the first things to do. After making an animation class, everything was so much easier and cleaner to deal with, and we ran into way fewer random bugs caused by lines that were coded incorrectly or were missing.

Scott:

The lesson I learned this year was going back and making code more efficient after making it work. While no set lesson is made for this, the skill is very good to have. It saves time later when going back and trying to find a certain line or helps put things together so that the code is more condensed and easier to read and use. While I found it hard to change things while still being able to make them work, I was able to put the entire animation class under one class instead of having it in two classes as what we originally had to make everything work. This extra class was not needed, so I went in and put everything into one class, thereby shortening the code and condensing things into one class to make sure it was all easier to find.