**COMP 3609, Game Programming**

**Lab of 2019-02-26**

**BatBallGame-Disappear2**

In this program, the image of the sprite disappears slowly until it is no longer visible. Then, it reappears and the process repeats itself. The disappearing effect is achieved by modifying each pixel’s *alpha* byte: initially it has a value of 255 and this value is reduced on each update by 5 until it becomes 0. When the alpha byte becomes zero, it is reset to 255. The pixel is completely opaque when the alpha byte has the value 255. it is completely transparent when the alpha byte has the value 0.

The edges of the image are transparent so whenever it is placed on a background, the background is normally shown instead of the edges:



However, when the program changes the alpha byte from 255 to 0, the image is displayed as follows:



As can be seen, the edges of the image are no longer transparent. How can the disappearing effect be achieved with the transparent edges staying transparent? In other words, how can the disappearing effect by achieved so that transparent pixels are not affected?

**BatBallGame-Disappear2-Fixed**

It should be noted that some of the pixels in the image are already transparent – namely, those that are on the edges. So, when the alpha value of all the pixels is set to 255 and then reduced to zero, the transparent pixels get an alpha value of 255 which is then reduced to zero. This causes the transparent pixels on the edges to become opaque, hence the image is seen as a rectangle.

This problem can be solved by getting the value of the alpha byte of every pixel before modifying it. This can be done in a similar manner to finding the red, green, and blue components:

pixelAlpha = (pixels[i] >> 24) & 255;

Before setting a pixel’s alpha value to the current alpha value (which is used in the *update()* method to update all the pixels), a check is made to determine if the pixel’s alpha value is zero. If so, it means that it is transparent; consequently, no modification is made to the pixel’s alpha value. If the pixel is not transparent, the current alpha value becomes the pixel’s alpha value:

if (pixelAlpha != 0) {

newValue = blue | (green << 8) | (red << 16) | (alpha << 24);

pixels[i] = newValue;

}

However, this causes another problem. When all the opaque pixels in the image become transparent, it means that the image will become totally transparent. There is now no way to distinguish the originally transparent pixels from the pixels which were originally opaque. The *if* condition will never become true since the alpha values of **all** the pixels are 0; consequently, the alpha values will never change from 0 and thus the image will stay totally transparent (i.e., invisible).

This new problem can be solved by making a copy of the sprite’s image when the current alpha value becomes less than or equal to 0 (in the *update()* method):

if (alpha <= 0) {

copy = copyImage(spriteImage);

alpha = 255;

}

Since *copy* is an exact copy of the sprite’s image, it starts off with the transparent and opaque pixels having their correct initial alpha values (instead of all the alpha values being 0).

Question: The program in this folder re-displays the image when the alpha value becomes 0 or less. The current alpha value is reset to 255 meaning that the opaque pixels will start off being completely opaque. Suppose we would like the image to slowly re-appear after it completely disappears. In other words, the current alpha value should go from 255 to 0 and then from 0 to 255. How, can this be achieved without making the originally transparent pixels opaque?