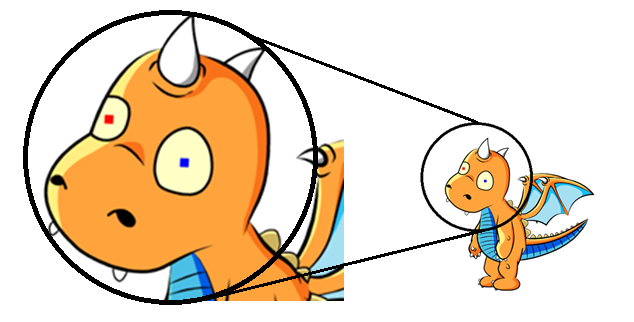
Eye Movements

One of the most interesting idea that we had was to make a creature whose eyes will track the cursor at all times. No matter what state of creature is or where the cursor is. The creature should always track the cursor. Initially we were animating eyes along with pupil in the Crazytalk Animator program. But mid way we realized that how would we implement the eye tracking. Problem is that the animation sprites we output already have eye pupil with it. That’s the eyes are fixed by the animation software itself. What we can do was to output animation without eyes. But then it would be practically impossible to know where to put the eye pupil.

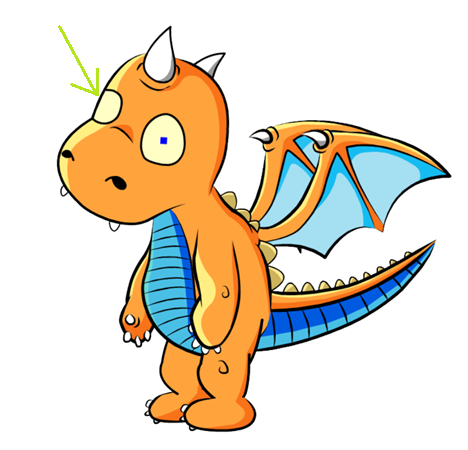
Eyes movements are most crucial part. Creature must at all times follow the cursor. To counter the problem, we followed following basic approach:

Output the animations without eyes as. Also output one set of eyes only and save it in resources folder. Then later when animation is required we would first load the creature on the screen followed by the pupil. But to do so we need to know exact location of eyes. Else there is no way do it correctly. What we did was innovative. Instead of output animations with or without eyes, we changed the eyes with two distinct colored square patches. The pupil was change to these patches. We selected red and blue color for this. Red for one eye and blue for other. Now you might be thinking how would this solve the problem.



The fact that the red and blue are the distinct colors as chosen carefully. We can somehow search for these colors. So what we did was to traverse each pixel of the each frame of the animation. Then we will compare it with the distinct red or blue pixel that we have put in the eyes. So we wrote a separate program for this. The program would search through all the frames(approx. 1000), and for each frame, it will go through all the pixels in the image looking for the distinctive color. On encountering the distinctive color, the program would do a Breadth First Search looking for similar pixels. For each of such pixels, the program will take average of their X and Y coordinates.

Now for each frame program will output the location of eyes to a separate file which holds the coordinates of the file for each animation sequence.

Doing so solves one problem. But still there few others left. Now that we have the location of the eyes, we don’t have the required animation sprites. The animations now have two square patch instead of blank eye. So what we did was to modify the previous search function. And set to replace every selected distinct colored red or blue pixel to the color of the background of the eye. Doing so will replace the red and blue pixels with the required empty space. Following diagram shows this on one eye(Image in the left):



So after processing both the eyes final frame would look as show above(Image in the right):

Doing so removes the square patches which were added earlier. We converted all the frames to form as shown above. And in return got the locations of eyes in each frame. Also the locations are in terms of pixels. But it doesn’t resolves the whole issue. We still need to get the original image back.

For getting the original frame back idea quite simple. First to output the base frame and the overlay it with eye bitmaps. Even though doing this gets the original frame back but still movement is not achieved. This just helps that the eyes are now actually floating over the frame. We still need to find out way so that eye follows the cursor.

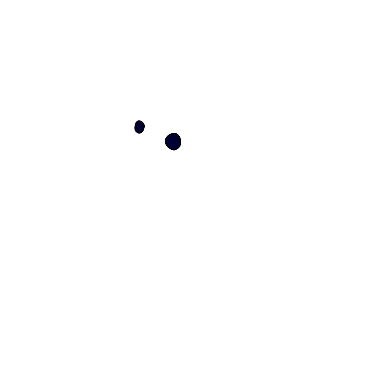
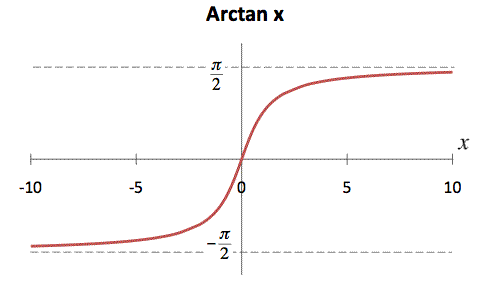




Diagram above shows the whole process in its flow chart form. We can see that initially frame had colored eyes but then later that frame gives off another frame and location of eyes. Then finally when we need to display the eyes we will just display them one over the another.

Now for the cursor tracking. So far we can display eyes separately any where over the frame. Now we need to add the tracking functionality. For this basic idea is to change the change in X and Y coordinates of the eyes according to the difference between the cursor and Dino. But this has its own problem. If we do that then the eyes will get out of the expected positions.

Instead what we want is that eyes should only move inside the eye sockets. And not the get out of them. So we basically required a transformation that transforms the given input range to restricted range. Also as eyes moves quickest when they are at center and then slows down as it approaches the eye borders. That something similar to the graph of sin function. But sin function has reversed S shaped covered around origin. That would serve the purpose but there is another better alternative. Inverse Tan function. Tan-1 has following graph:

Also the Arctan function is defined for (-inf, +inf) whereas the sin function changes its values after pi / 2. So arctan function solves the purpose. It perfectly simulates displacement of eye with respect to the distance between eyes and cursor. Now the actual transformation used is as follows:

*dX = distance(eye\_X, cursor\_X)*

*dY = distance(eye\_Y, cursor\_Y)*

*dX = dX / 100*

*dY = dY / 100*

*dX = arctan(dX)*

*dY = arctan(dY)*

Now instead to giving eye output at coordinates (eye\_X, eye\_Y) output the eyes at (eye\_X + dX, eyeY + dY). This works perfectly well. Though it is not the form that we have used. Eyes movements are more defined using more number of transformations. Those are complex and requires more visual representations to be shown.

**Different Eye Positions Using Above Transformation**