1. Inteligencia Artificial

Dentro de la Inteligencia Artificial que hemos dotado a nuestro juego, vamos a desarrollar nuestros dos principales funcionalidades usadas. Las cuales son:

* Visión artificial del enemigo
* Maquina de estados del enemigo
  1. Visión del Enemigo

Para la implementación de la visión artificial del enemigo hemos usado una funcionalidad proporcionada por nuestro motor usado (OGRE) llamada ***RenderTargetListener***, la cual nos permite recibir notificaciones por eventos relacionados con el objeto ***RenderTarget***. El objeto RenderTarget puede ser desde una escena, una pantalla o una textura, que para nosotros era lo que nos interesaba, ya que nosotros necesitábamos un ***Render a Textura***. Vamos a explicar ahora que es eso de Render a Textura.

* + 1. Render a Textura

Antes de explicar que es el render a textura, vamos a dar un contexto a su uso.

**¿Por qué usamos ésta técnica?**

Esta técnica nos proporciona un mecanismo de visión artificial fácil de implementar, ya que consiste en hacer una “foto” de la escena que estaría viendo nuestro enemigo y después interpretar ésta foto para determinar si el enemigo está viendo a nuestro héroe o no.

**¿Como se ha implementado ésta técnica?**

Vamos por partes; usando una cámara que colocamos delante de nuestro enemigo tenemos una visión panorámica de lo que tiene delante (Ver figura 1a):

Figura 1a

Nos creamos una textura “manual”, de la cual usamos su objeto interno *render a textura* para asignarle nuestra cámara. De manera, que ahora tenemos vinculada la cámara a éste objeto. Posteriormente, a éste objeto render a textura le añadimos como listener, nuestro RenderTargetListener, de manera que cada vez que se renderiza un frame obtenido por la cámara nos pasa a nuestro listener para que nosotros lo tratemos. Y ahora viene lo mejor, ya que con éste listener que se ejecuta a cada frame, nosotros realizamos el siguiente proceso:

1º Antes de que se calcule el frame (con nuestro listener tenemos dos callbacks antes y después de calcular el frame de la escena). Nosotros tenemos dos objetos en el héroe, uno visible (el modelo de nuestro héroe) y otro invisible (un cubo blanco); el objeto invisible esta adjunto al héroe, de manera que cualquier movimiento asociado al modelo del héroe aplica al cubo. Pues bien, nosotros ponemos a invisible el modelo del héroe y a visible nuestro cubo completamente blanco.

2º Hacemos una foto de éste frame mediante el uso del objeto textura, el cual nos proporciona un método para volcar ésta a una imagen.

3º Con la imagen obtenida muestreamos los píxeles, de manera que aquellos píxeles que son blancos los contamos. Y mediante el uso de un umbral, tenemos nuestro “*detector de héroes*”. (Ver figura 1b).

Figura 1b

* 1. Máquina de Estados

Para la máquina de estados usada para nuestros enemigos hemos tenido en cuenta los siguientes estados:

Cada uno de nuestros enemigos implementa una pequeña máquina de estados, de manera que en función del estado en que se encuentre obrará de una manera u otra.

Vamos a desglosar cada uno de los estados y lo que implica estar en ellos.

* + 1. Estado WATCHING

Este estado es el estado “natural” de nuestro enemigo, es decir, nuestro enemigo, mientras no vea a nuestro héroe, irá por nuestro escenario siguiendo una ***ruta*** (Ver apartado de Rutas del Enemigo) vigilando por si aparece nuestro héroe.

En el momento de que nuestro enemigo aviste al héroe, pasaríamos al estado ***ALERT***. Además, en éste caso, almacenaremos la posición donde vimos al enemigo para poder usarla en el siguiente estado.

* + 1. Estado ALERT

En éste estado quiere decir que nuestro enemigo a entrado en modo alerta, ya que nos ha visto y por lo tanto, si una vez que estamos dentro de éste estado persistimos dentro de su campo de visión pasaríamos al estado ***SHOOTING***.

En caso de habernos perdido de vista durante éste estado, pueden ocurrir dos cosas:

* Si nos ha perdido de visto antes de unos 2 segundos pasa a modo WATCHING, ya que no ha sido tiempo suficiente como para asegurar que lo que hemos visto sea el heroe.
* Pero si nos ha perdido de vista despues de haber permanecido delante de él más de éstos 2 segundos, entonces pasa al estado ***CHASING***. Almacenando el instante temporal del momento en cuanto arrancamos la persecución.
  + 1. Estado SHOOTING

Al pasar a éste estado, pueden ocurrir dos cosas:

* Que el enemigo nos dispare, ya que nos mantenemos en su campo de visión, de manera que si no estuviésemos orientados hacia nuestro héroe, reorientaríamos a nuestro enemigo hacia él para que el disparo fuese lo más certero posible. También entre cada disparo controlamos que no supere un tiempo inferior a 2 segundos entre cada uno.
* Si estamos en éste estado y desaparecemos de su campo de visión, entonces pasa al estado CHASING. Almacenando el instante temporal del momento de iniciar la persecución.
  + 1. Estado CHASING

En éste estado, partiendo de la premisa de que si hemos entrado en éste estado es porque inicialmente nos vio el enemigo y hemos desaparecido de su campo de visión; tenemos las siguientes casuísticas:

* Que el enemigo nos vea y por lo tanto, si persistimos en su campo de visión pasamos al estado ALERT.
* Que el enemigo no nos vea, pero:
  1. Puede que no haya llegado al punto donde nos vio por ultima vez, así que le hacemos ir hasta el punto último donde nos vio. Como éste desplazamiento provoca que nuestro enemigo se salga de la ruta, realizamos un almacenamiento de puntos para poder usar este conjunto de puntos, como puntos de retorno a la ruta original. Ver la ***Nota Adicional*** al pie para más información.
  2. Si estamos en el punto último donde nos avistó, nos detendremos y entonces realizaremos un barrido de 360º en busca del héroe, por si lo podemos volver a ver.
  3. Si después de haber realizado el barrido de 360º no vemos a nuestro héroe, usando nuestro conjunto de puntos almacenados como puntos de retorno a la ruta original, volveremos a la ruta y cambiaremos al estado WATCHING, para que prosiga por la ruta.

**Nota Adicional**

Puede ocurrir que los enemigos en el momento de iniciar la persecución vayan por zonas del escenario que luego, por si solo, no sabría como obrar para volver a su ruta original. Para ésto, se implementó el almacenamiento de puntos de recuperación (a modo de “migas de pan”), los cuales se van almacenando cada poco tiempo y con cada desplazamiento del enemigo por el escenario mientras la persecución persista.