Inteligencia Artificial para Videojuegos – Capítulo 1.5

Introducción

Los árboles de comportamiento (Behavior Trees, BT de ahora en adelante) se han convertidoenuna herramienta muy popular para modelar la inteligencia de personajes en videojuegos. Halo2fueuno de los primeros videojuegos que describió en detalle el uso de los árboles y las ventajas aportadas por lo que se lo considera el pionero en la aplicación de BT a videojuegos. Como consecuencia gran cantidad de estudios siguieron los pasos y se apropiaron de los BT. En esta unidad vamos a realizar un introducción al modelado de comportamientos usando BT y veremos algunos ejemplos.

Behavior Trees

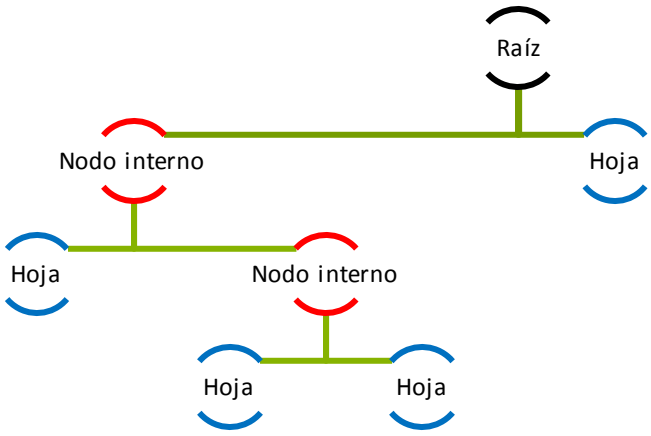
¿Porqué?

Los BT poseen ciertas características que los hacen muy populares, entre las más destacadas podemos mencionar:

- Separan el diseño de las estrategias de IA de su implementación. Esto permite que puedan realizar el diseño personas que no saben programar.
- Permiten atacar problemas dónde las máquinas de estado presentan problemas y complicaciones innecesarias.
- Es posible reutilizar en gran medida los árboles, para componer comportamientos más complejos.
- Están de moda.

Estructura

Un BT es una organización jerárquica de elementos denominados tareas. La estructura utilizada es un árbol (de allí proviene su nombre). A continuación se muestra un ejemplo de dicha estructura:



En próximas unidades estudiaremos en mayor detalle que es un árbol, por ahora simplemente diremos que es un conjunto de nodos conectados por aristas. El nodo que no tiene padre se

denomina raíz y se encuentra dibujado en negro. El resto de los nodos pueden internos u hojas. Los internos son aquellos que poseen hijos, mientras que las hojas son los nodos que no poseen ningún hijo. Es decir, los que están más abajo en la jerarquía.

Tareas

A diferencia de las máquinas de estado, cuyo bloque constitutivo es el estado, el bloque constitutivo de un BT es una tarea. A cada tarea se le da un tiempo de CPU para su ejecución y cuando termina devuelve un código de estado. Este código indica si la tarea pudo terminar su ejecución correctamente o no (Éxito o Fracaso). Además algunos desarrolladores agregan estados extra de retorno que permiten un mayor control sobre la ejecución. En el árbol cada tarea es un nodo, y dependiendo del tipo de tarea, será un nodo hoja o un nodo interno.

La combinación de tareas permite lograr comportamientos realmente "inteligentes", y es aquí donde reside el gran potencial de los BT. Las tareas las podemos clasificar en varios tipos y dependiendo de la implementación podemos ver algunos extras. En esta discusión teórica vamos a estudiar los más básicos que generalmente siempre se encuentran. Los tipos de tarea pueden ser:

Condiciones y acciones

Las condiciones chequean alguna condición sobre el juego. Por ejemplo si el personaje X está cerca del personaje Y, o si el enemigo X tiene el escudo destruido, etc. Las tareas de tipo condición devuelven verdadero si la condición se cumple y falso en caso contrario.

Las acciones por otro lado son tareas que alteran el estado del juego. Aquí es donde efectivamente se realizan las tareas tales como indicar al personaje que se mueva, reproducir algún sonido, modificar una animación, enviar un mensaje a otro objeto, etc.

Tanto las condiciones como las acciones son nodos de tipo hoja en el árbol. Por ejemplo:

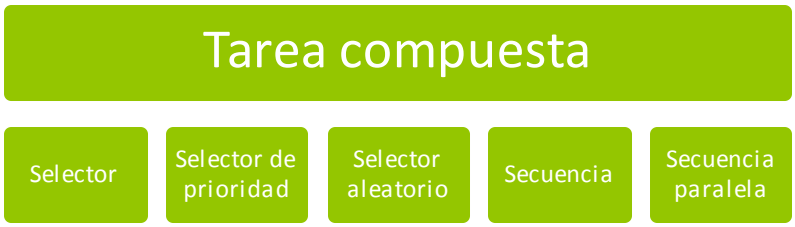
En este árbol tenemos un ejemplo muy sencillo con dos tareas. La primera tarea es de tipo condición y se encarga de chequear si el enemigo sigue vivo. La segunda tarea en cambio es una acción que

hace que el personaje escape. Cuando uno desarrolla utilizando BT lo que implementa son las condiciones y las acciones.

Tareas compuestas

El tercer tipo de tareas que habíamos mencionado son las compuestas. Las tareas compuestas son nodos internos del árbol, es decir que poseen como hijas tareas subordinadas. Estas tareas subordinadas pueden ser de acciones, condiciones o compuestas.

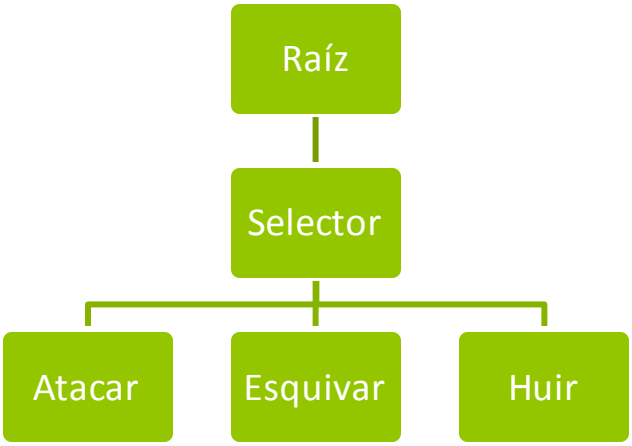
Las tareas compuestas pueden ser de distinto tipo



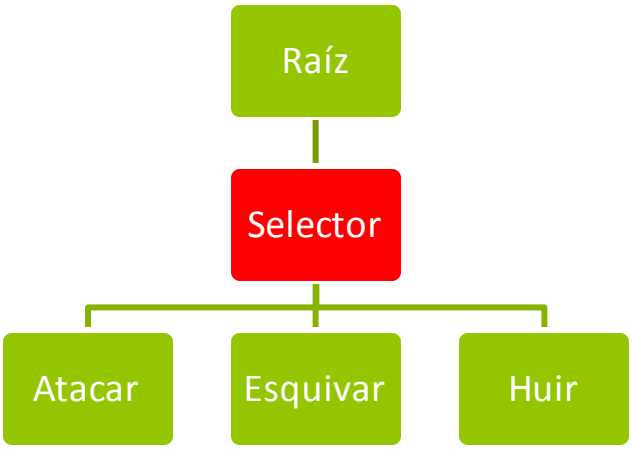
Selector

La tarea de tipo selector ejecuta secuencialmente las tareas hijas hasta que alguna devuelve un código de éxito. Es decir ejecuta los nodos hijos hasta que alguno se complete exitosamente. Si se da esta condición entonces el selector devuelve a su nodo padre un código indicando el éxito. Si en cambio ninguna de las tareas se completaron correctamente, entonces devuelve un código de fracaso.

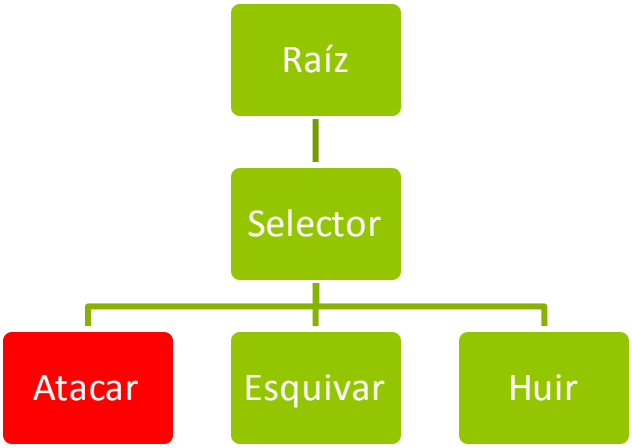
Veamos un ejemplo para entender mejor como funciona, si tenemos el siguiente árbol:



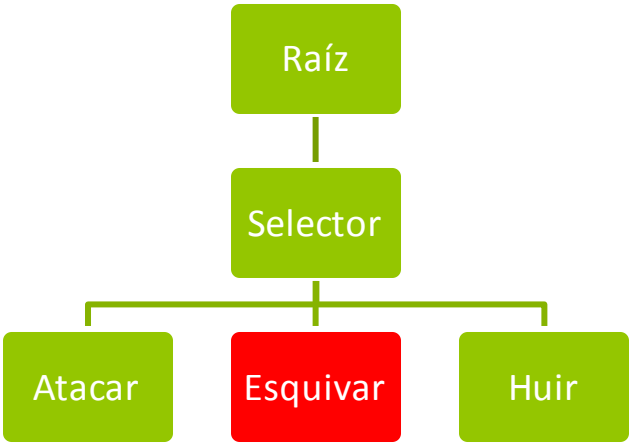
El siguiente árbol describe un personaje que entra en combate. Este árbol se ejecutará cada vez que se actualice la IA del personaje. Cada vez que se ejecuta comienza desde la raíz. En este caso lo primero que hay es un selector, por lo que pasa en primera instancia a ejecutar el selector:



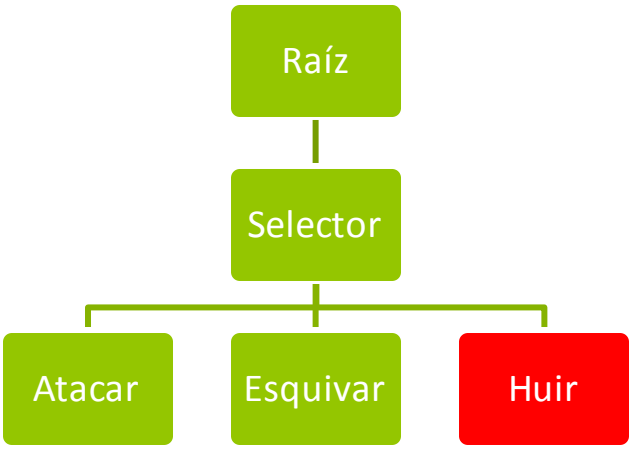
El selector en este caso primero intentará ejecutar la acción atacar:



Si el personaje puede atacar exitosamente entonces devuelve el estado “success” al padre y el selector a su vez devuelve “success” a la raíz y se termina el ciclo de IA. En cambio si el personaje no puede atacar por algún motivo, la tarea devovlerá “failure” y el selector al recibir dicho estado intentará ejecutar el próximo hijo:



Esta vez el personaje intentará esquivar algún golpe o disparo. Sino tiene éxito devolverá "failure" y el selector padre intentará ejecutar la última tarea en su lista:



En caso de no tener éxito entonces devolverá "failure" y el selector al no tener más nodos hijos devolverá "failure" al padre.

Selector de prioridad

Existe un tipo especial de selectores denominados selectores de prioridad. Funcionan de manera similar a los selectores comunes, pero permiten cambiar el orden en que se evalúan los hijos en función de algún criterio de prioridad asignado por el desarrollador.

Selector aleatorio

Como su nombre lo dice, funciona igual que el selector común, pero el orden en que ejecuta los hijos es aleatorio.

Secuencia

La secuencia es muy similar al selector, pero en este no se ejecuta hasta encontrar una que devuelva "success", sino que se ejecuta **mientras** las tareas devuelvan "success". Es decir busca que se completen todas exitosamente. Si esto sucede la secuencia devuelve "success" al padre. Si alguna falla, la secuencia se detiene y devuelve "failure" al padre.

Secuencia paralela

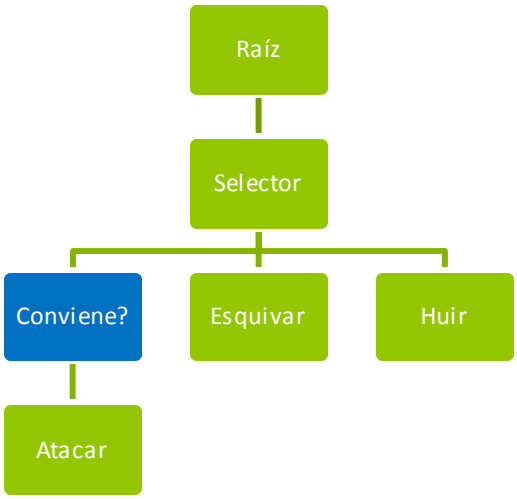
La secuencia paralela ejecuta todos los nodos hijos independientemente del valor de retorno. Si fallan al menos n hijos entonces la secuencia devolverá "failure". En caso contrario, es decir si fallan menos de n hijos entonces la secuencia devolverá "success". El valor n es definido por el usuario cuando diseña el árbol.

Decoradores

Los decoradores son un tipo especial de tarea que poseen un solo hijo y modifican su comportamiento de alguna manera.

Un tipo común de decorador es aquél que se conoce como filtro y su función es evaluar alguna condición y en base al resultado decidir si el hijo debe ejecutarse o no. En el siguiente árbol se

puede observar una pequeña modificación del ejemplo anterior. En el mismo se agregó un decorador (azul). Dicho decorador evalúa alguna condición y si se cumple entonces ejecuta el hijo. En este caso el código que devuelve el decorador al padre es el que recibe del hijo. Si en cambio la condición no se cumple el decorador devuelve "failure" al padre.



En este caso el decorador debería evaluar si conviene atacar al objetivo o no. Si bien es cierto que este mismo comportamiento se podría haber obtenido con condiciones, el ejemplo es simplemente ilustrativo.

Existen varios tipos de decoradores más, pero depende en gran medida de la biblioteca que utilicemos.

Bibliotecas

La verdadera utilidad de los árboles de comportamiento se aprecia cuando se dispone de un diseñador que permita diseñar los árboles y que genere automáticamente la conexión necesaria con el código.

Al ser algo relativamente nuevo no existen muchas bibliotecas disponibles. Para C++ pueden descargar dilib (http://dilib.dimutu.com/). Para Unity3d hay una muy popular llamada behave. En la plataforma podrán encontrar un video dónde se hace un pequeño tour sobre su funcionalidad.