Proyecto: Trails

https://github.com/amprubio/Trails

Fecha del enunciado: 25 de mayo de 2021
Fecha de defensa: 15 de junio de 2021
Fecha de entrega: 16 de junio de 2021

1. Introducción

Los Behaviour trees son la tecnología utilizada hoy en día en la industria para implementar comportamientos complejos en los NPCs. Los BTs al igual que las SM se apoyan en acciones y elementos de percepción básicos programados directamente en el lenguaje del motor (con frecuencia C#/C++), y permiten combinarlos para generar comportamientos complejos.

En general, los BTs se pueden percibir como una evolución de las máquinas de estados, aunque su expresividad es equivalente a la de un lenguaje de programación de propósito general, por ello, los BTs se pueden construir utilizando lenguajes visuales, y si su uso se restringe a especificar aspectos concretos del juego, pueden ser más adecuados para los diseñadores que las propias máquinas de estados. De hecho, en Halo 2, el juego que popularizo el uso de BTs, estos se usaban tanto por programadores como por diseñadores cuya mayoría también sabían programar.

Los Behaviour Trees o BTs son utilizados en videojuegos para modelar la inteligencia artificial de los NPCs de una forma similar a las SM. La diferencia principal radica en que los BTs eliminan las transiciones de las SM, lo que permite subsanar los problemas derivados del crecimiento del número de transiciones típicos de las SMs. De esta forma, los arboles de comportamiento definen un mecanismo para decidir cuál es la siguiente acción que se ejecutará. Para ello, se incorpora información adicional de control a las acciones que determine cuál de ellas debe ser ejecutada en su momento.

Esa información de control es modelada normalmente mediante nodos intermedios que toman decisiones sobre como y en qué orden es en el que deben ejecutarse las acciones. Estas decisiones se toman utilizando los datos del entorno mediante nodos especiales que los chequean y utilizando el valor de retorno de las acciones. Y eso porque, al contrario que en una FSM, cuando la ejecución de un nodo de un BT termina, este notifica a su nodo padre si la ejecución de la acción tuvo éxito (Success) o no (Failure). Con esa información de finalización y la información de entorno, los nodos intermedios toman decisiones sobre cuál es la siguiente acción a ejecutar.

Los nodos de los BTs se distinguen entre los nodos hoja que contienen las acciones que alteran el entorno y los nodos internos, o nodos de decisión, con los que se montan los comportamientos complejos agrupando acciones. Estos últimos se agrupan en diferentes tipos

según sus reglas de funcionamiento y existen distintas colecciones de tipos de nodos que proporcionan expresividad equivalente. Algunos de los más utilizados son:

1- Composites: Estos nodos se caracterizan por tener una serie de hijos en su conjunto y su resultado depende del resultado de sus hijos. Estos se pueden categorizar en:

– **Secuencias**: Los hijos se ejecutan en el orden que se han definido. Cuando el comportamiento que esta ejecutandose actualmente termina con éxito, se ejecuta el siguiente. Si termina con fallo, este también lo hace. Si todos se han ejecutado con éxito, el nodo devuelve éxito.

– **Selector**: los ejecuta en orden hasta encontrar uno que tiene exito al contrario que la secuencia. Si no encuentra ninguno, termina con fallo. El orden de los hijos del selector proporciona el orden en el que se evaluan los comportamientos.

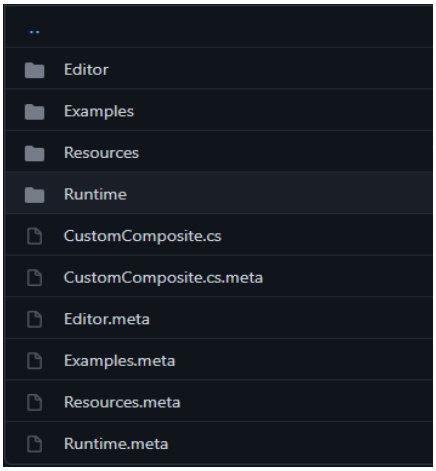
– **Parallel**: el nodo parallel ejecuta a la vez todos los comportamientos hijo del nodo mediante corrutinas. No necesariamente esta ejecución debe ser concurrente, puede ser uno detrás de otro, pero dentro de la misma iteración en el bucle de juego. Los nodos parallels pueden tener diferentes políticas de finalización. Pueden finalizar como una secuencia, es decir en el momento que un nodo falle todo el nodo parallel falla, o como un selector, hasta que todos los nodos han fallado no falla el nodo.

2- Decorators: Los decoradores son nodos especiales monoparentales y que permiten modificar el comportamiento o el resultado de ese hijo, incluyendo, guardas que controlan si el nodo se ejecutará o iteradores que permiten especificar un número de ejecuciones (Repeaters).

3-Actions: Son nodos sin hijos que ejecutan una accion en concreto, su resultado depende de la acción en sí misma, un ejemplo de ella es mover un objeto a una posición x, devolverá success cuando se alcance dicha posición.

2. Implementacion

El proyecto está esturcturado en diferentes directorios.



Dentro de la carpeta de Trails tenemos el source y en el encontramos 4 carpetas diferentes.

En la carpeta de Editro se encuentra todo lo relativo a la representación gráfica del plugin en sí.

En Examples encontraremos en un futuro una escena de ejemplo que servirá como guía para el uso e implementación de los árboles de comportamiento.

En la carpeta Resources encontramos todos los recursos graficos como las texturas asi como los estilos de GUIStyles que dotan de representación gráfica a una fuente.

Finalmente en Runtime nos encontramos con toda la implementación de los Nodos, estructuras arboreas para su organización, el comportamiento de los nodos, los recursos necesarios para leer y crear los archivos .asset

Todos los comportamientos heredan de BehaviourNode para una implementación uniforme.

Las clases padres de comportamientos son a efectos prácticos clases virtuales, no se puede crear una Acción, de debe crear un componente que herede de Acción como puede ser el Wait.

BehaviourNode:

Ofrece la funcionalidad común a todos los comportamientos.

Ayuda a su representación y ejecución.

Status: Failure, Success, Running y None

Sirven para que el editor maneje la ejecución de los nodos.

mUniqueID_, mName_, mPos_

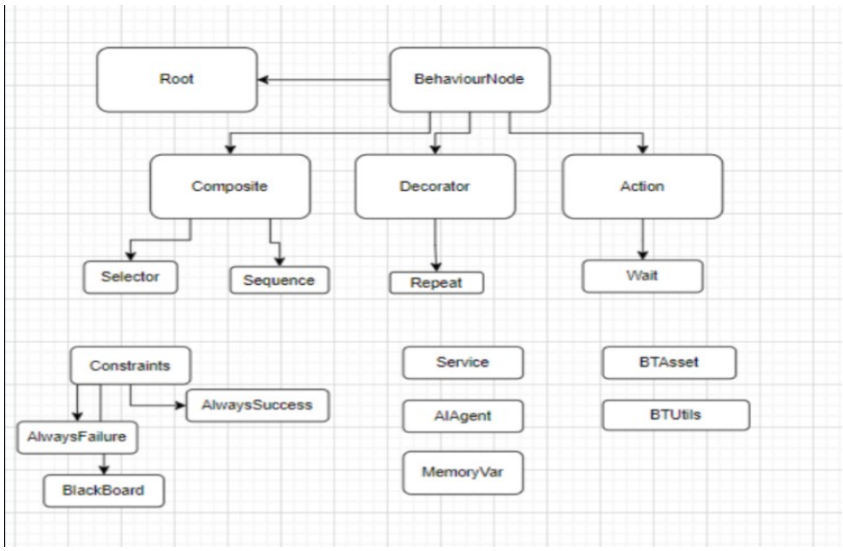
Contiene información sobre el nodo, su representación e indentificación.

mComment_

Contiene información del comportamiento que se mostrará al pasar el ratón sobre él.

Los behaviourNodes están compuestos de métodos virtuales que implementarán los componentes y que controlan los estados del nodo.

OnStart, OnReset, Run, OnEnter y OnExit.



Action - Comportamientos simples.

Wait

Usa un temporizador interno para controlar el tiempo que se ha ejecutado y mientras este no llegue a 0 el nodo devolverá Running (Se ejecutará el nodo), Cuando se agota cambiará el estado a Success permitiendo continuar con los comportamientos que los sigan.

Composites - Acciones compuestas

Parallel

Permite la ejecución de multiples acciones simultáneas.

RandomSelector

Elige entre varias acciones para ejecutarla.

Selector

Ejecuta una u otra accion basandose en una condición.

Sequence

Ejecuta acciones en secuencia (Una detrás de otra esperando a su acción anterior).

Constraints - Condiciones para una acción

AlwaysFailure

Nodo que siempre devuelve Success

AlwaysSuccess

Nodo que siempre devuelve Failure

BlackboardConstraint

Dado un valor A y un valor B, si son del mismo tipo podemos compararlos.

Decorators - Comportamientos que modifican la acción de su hijo

NodeGroup

Ejecuta el hijo asignado y devuelve Success indiferentemente del resultado del hijo.

RepeatForever

Repite el mismo comportamiento indefinidamente.

Repeater

Repite un comportamiento hasta que se cumple un número de ejecuciones.

Editor

En cuanto al editor, el cual no se ha podido completar, pero nos encontramoos con una

representación gráfica del core más algunos añadidos para manejar eventos y crear elementos más relacionados a interfaces de usuario.

-**BehaviourTree**: clase principal en la cual se representa la ventana, contiene referencias a otras clases que manejan eventos y dibujan texturas en ella.

- **BTEditorGrid**: clase que dibuja el fondo de a pantalla.

- **BTEditorCanvas:** clase que maneja los eventos de la ventana.

- **BTEditorGraph:** clase que representa el Behaviour Tree, asi como sus eventos

- **BTEditorGraphNode:** clase que representa cada uno de los nodos en pantalla así como sus posibles hijos, en caso de tenerlos.

- **BTEditorStyles:** Contiene información de las texturas, fuentes y propiedades que usamos para representar todos los elementos de la ventana

- **BTEditorTreeLayout:** Enumerado que ayuda a configurar el layout a trabajar (Vista horizontal, vertical o libre)

- **BTScriptCreation:** Contiene todo lo necesario para crear nuevos scripts con la herencia de los nodos correspondientes.

3. Referencias y ampliaciones

- Ian Millington IA for Games 3rd Edition
- Documentación sobre el editor de Unity
  https://docs.unity3d.com/ScriptReference/GUILayout.html
- Repositorios:
  https://github.com/Tencent/behaviac
  https://github.com/yoshidan/UniBT
  https://github.com/end1220/BehaviourTree