Guía de uso de la herramienta de Unity

1. Como importar y configurar el paquete de Unity

El paquete "Behaviour API Unity Tool" incluye la API de creación de sistemas de comportamiento además de componentes para crear, ejecutar y depurar estos sistemas dentro de Unity.

Para usarlo se debe importar el paquete dentro de la carpeta "*Assets*". Si el paquete se importa en otra carpeta, se debe especificar la ruta en la configuración.

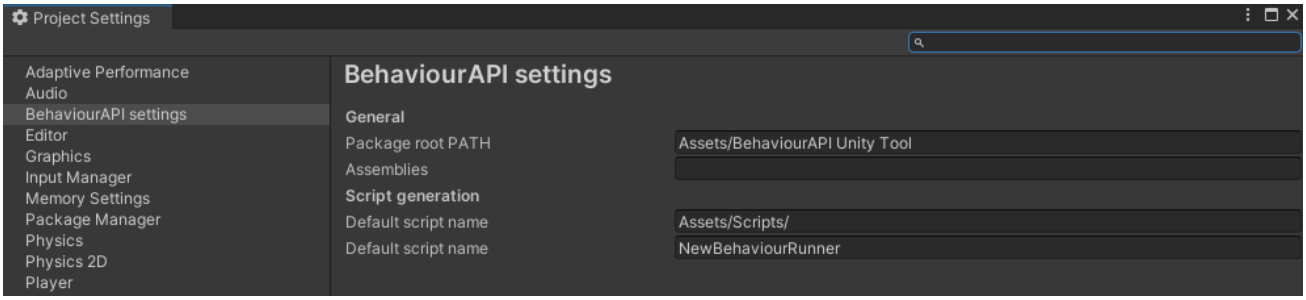
1.1 Contenido

El paquete se divide en varias partes:

- **Libraries**: Contiene las librerías de la API de C# compiladas.
- **Framework**: Todo el código interno para adaptar las librerías a Unity.
- **Runtime**: Incluye los componentes de Unity y otros elementos destinados a ser usados por el programador.
- **Editor**: Código y assets para la herramienta del editor de sistemas.
- **Demo**: Varias escenas con demos de ejemplo.

1.2 Configuración

Algunos elementos de la herramienta se pueden configurar en *Proyect Settings > BehaviourAPI Settings*.



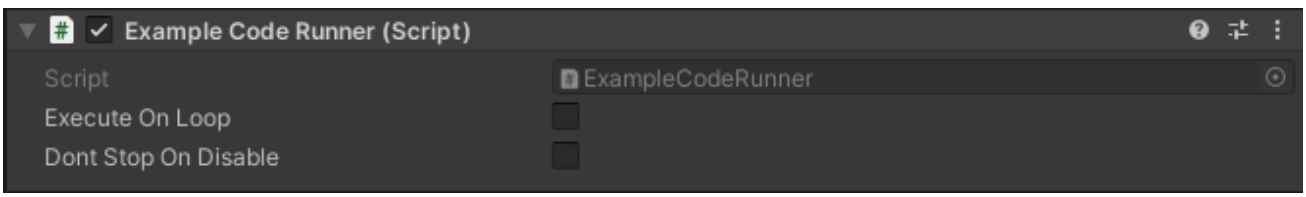
1 Menú de configuración

- **Package root Path**: Indica el directorio raíz donde está el paquete de Unity.
- **Assemblies**: Si el proyecto usa *Assembly Definition Files*, es necesario añadir el nombre de los ensamblados en este campo, separados por el carácter ";" (Si incluyen tipos de nodos, acciones o percepciones creadas por el usuario).
- **Default script path**: El directorio por defecto donde se generan los scripts.
- **Default script name**: El nombre por defecto de los scripts generados.

2. Crear sistemas de comportamiento en Unity

2.1 Como funcionan los scripts

La herramienta incluye varios scripts para crear, usar y depurar sistemas de comportamiento en *GameObjects* de unity. Estos scripts heredan de la clase `BehaviourRunner` y usan los eventos de Unity para crear y ejecutar el sistema de comportamiento:



2 Inspector de los scripts que heredan de BehaviourRunner

- `Awake`: Se crea el sistema de comportamiento
- `Start`: Se lanza el método Start del grafo principal.
- `Update`: Se lanza el método Update del grafo principal. Si se activa el flag ***ExecuteOnLoop*** el grafo se ejecutará en bucle.
- `OnDisable`/`OnEnable`: Se lanzan los métodos Stop y Start respectivamente solo si el flag ***Don't Stop On Disable*** no está activo. Esto permite decidir si cuando el componente se desactive y vuelva a activar se reiniciará la ejecución del grafo o solo se pausará.

Importante: Como estos eventos están implementados en la clase base no se pueden sobreescribir en las subclases. En su lugar se pueden sobreescribir los métodos `OnAwake`, `OnStart`, `OnUpdate`, `OnEnableSystem` y `OnDisableSystem` (Incluyendo siempre la llamada al método de la clase base).

```
protected override void OnAwake()
{
    // Código anterior a crear el sistema de comportamiento
    base.OnAwake();
    // Código posterior a crear el sistema de comportamiento
}
```

2.2 Contexto de ejecución de Unity

Los scripts de sistemas de comportamiento usan la funcionalidad de **pasar un contexto de ejecución** para que las acciones y percepciones tengan una referencia al gameobject que está ejecutando el grafo y a sus componentes principales, a través de la clase `UnityExecutionContext`.

Importante: Como estos componentes ya lo implementan de base, no se debe usar el método SetContext en los scripts que hereden de ellos.

En los ejemplos de UnityActions y UnityPerceptions que incluye el paquete se puede ver como usar esta funcionalidad.

2.3 Crear un sistema de comportamiento por código

Sirve para crear un sistema de comportamiento por código. Para ello se crea una clase que herede de ***CodeBehaviourRunner*** y se sobreescribe el método *CreateGraph* para crear los grafos. Al final del método se devuelve el grafo principal.

```
public class ExampleRunner : CodeBehaviourRunner
{
    protected override BehaviourGraph CreateGraph()
    {
          var graph = …;
          // Crear nodos, acciones, etc.
          return graph;
    }
}
```

2.4 Crear un sistema de comportamiento con el editor

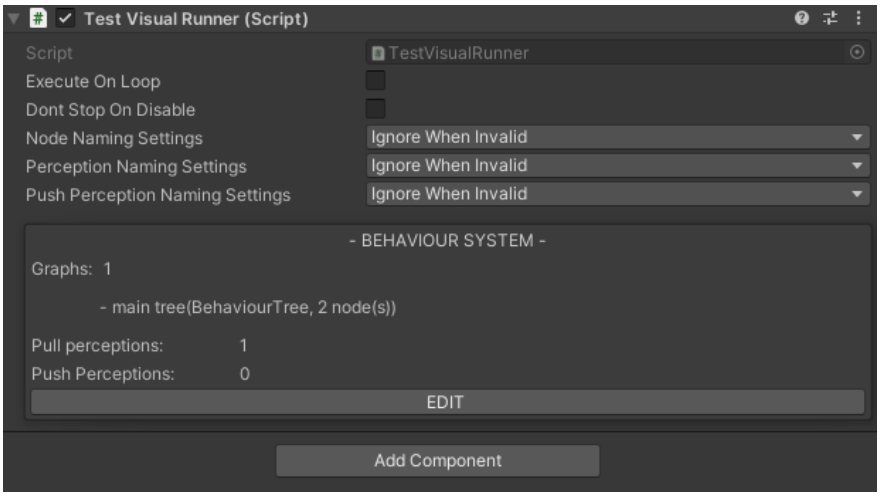
Crea un sistema de comportamiento usando la ventana del editor, creando una clase que herede de `EditorBehaviourRunner`.

Esta clase incluye métodos para acceder con código a los grafos, percepciones pull y percepciones push creadas con el editor. Para ello se usan los métodos `FindGraph`, `FindPerception` y `FindPushPerception` y se pasa como parámetro el nombre del elemento. Para buscar un grafo o percepción de un tipo específico se

usan los métodos genéricos `FindGraph<T>` o `FindPerception<T>`. Para que no se lance excepción si no se encuentra el elemento se usan los métodos `FindGraphOrDefault`, `FindPerceptionOrDefault`, etc.

Para poder buscar estos elementos el script crea diccionarios internos y desde el inspector del script se puede configurar como se guardan cada uno de los elementos en estos diccionarios.

- *Ignore Always*: No se guardan nunca.
- *Ignore when invalid*: Solo se añade si el nombre es válido.
- *Try add allways*: Se intenta añadir siempre y lanza una excepción si dos nombres coinciden.



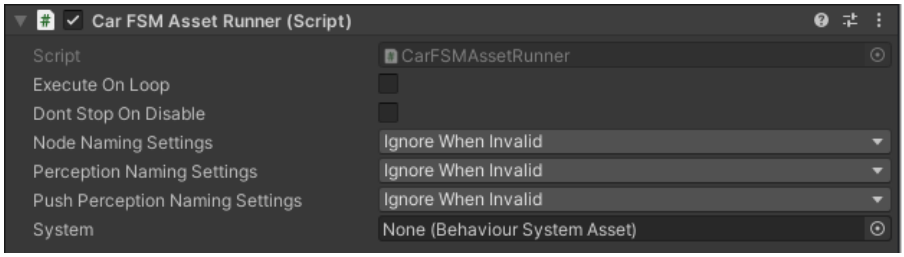
3 Inspector de los scripts que heredan de EditorBehaviourRunner

Es posible modificar el sistema de comportamiento creado en el método virtual *ModifyGraphs*. Esto puede usarse para asignar acciones, percepciones, guardar referencias a nodos, etc.

```
protected override void ModifyGraphs()
{
   var node = FindGraph("main tree").FindNode<LeafNode>("leaf 1);
   node.Action = new FunctionalAction(…);
}
```

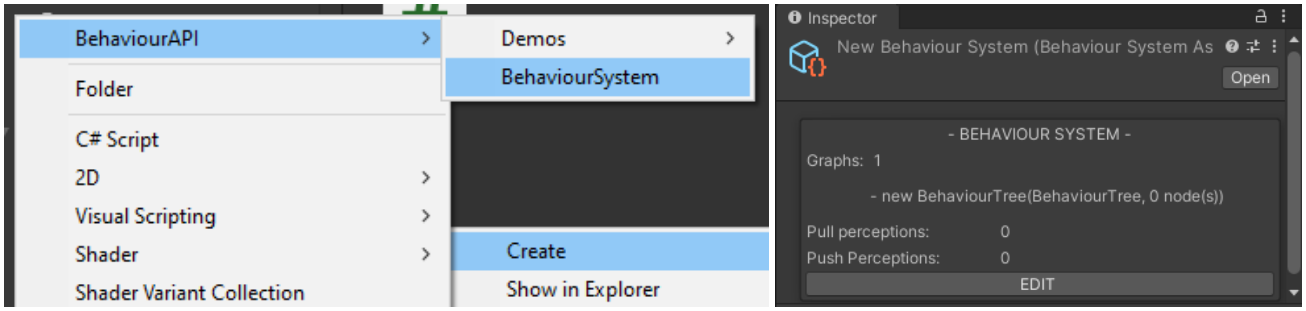
2.5 Crear un sistema de comportamiento reutilizable

Es posible ejecutar un sistema de comportamiento guardado como un asset en los archivos del proyecto usando el componente `AssetBehaviourRunner`. Esto permite ejecutar un mismo sistema de comportamiento en varios componentes distintos.



4 Inspector de los scripts que heredan de AssetBehaviourRunner

Para crear un sistema de comportamiento y guardarlo como un asset del proyecto se hace clic derecho en la ventana de proyecto y se selecciona *Create > BehaviourAPI > BehaviourSystem*.

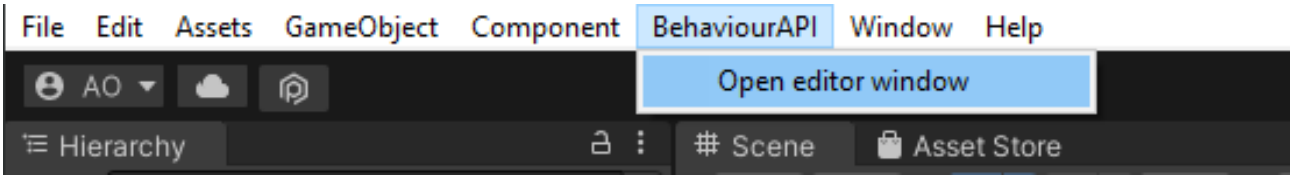


5 Pasos para crear un sistema de comportamiento como asset

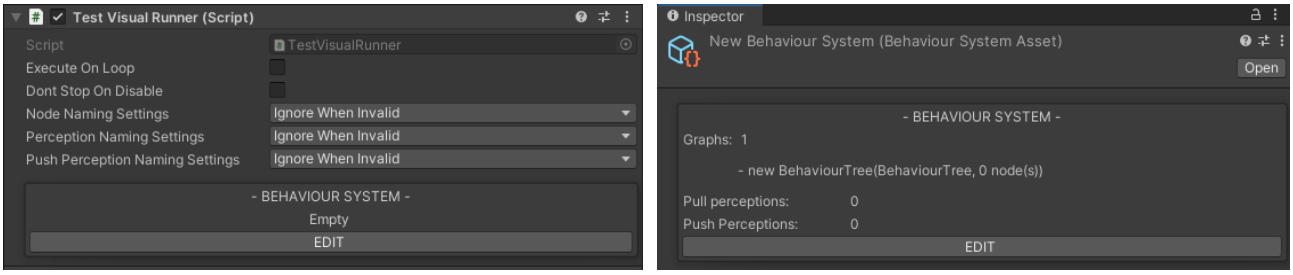
Para usarlo en un script se crea una clase que herede de `AssetBehaviourRunner` y se arrastra el asset al campo "*System*".

3. Guía para usar la ventana del editor de sistemas de comportamiento

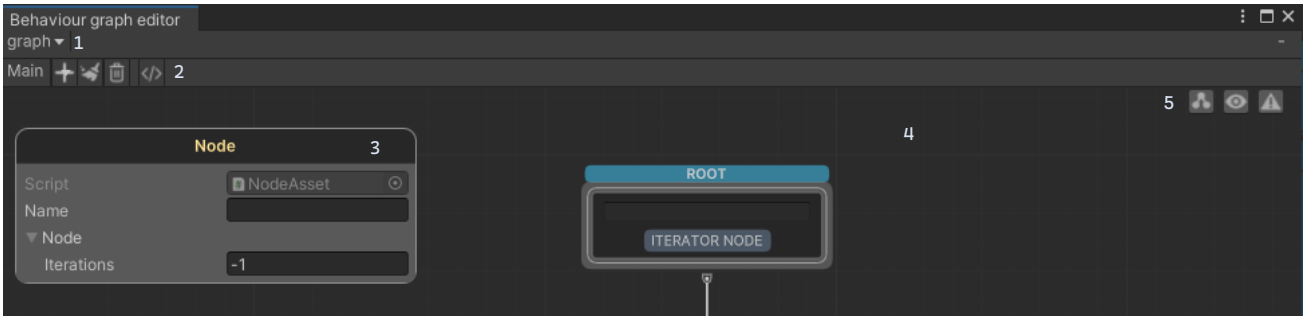
La ventana del editor puede abrirse desde el menú *BehaviourAPI > Open Editor Window*. Para editar un sistema de comportamiento se debe pulsar el botón *Edit* en un componente de tipo `EditorBehaviourRunner` o en un asset de tipo `BehaviourSystem`.



6 Como abrir la ventana del editor



7 Botones para editar un sistema de comportamiento



8 Interfaz de la ventana del editor

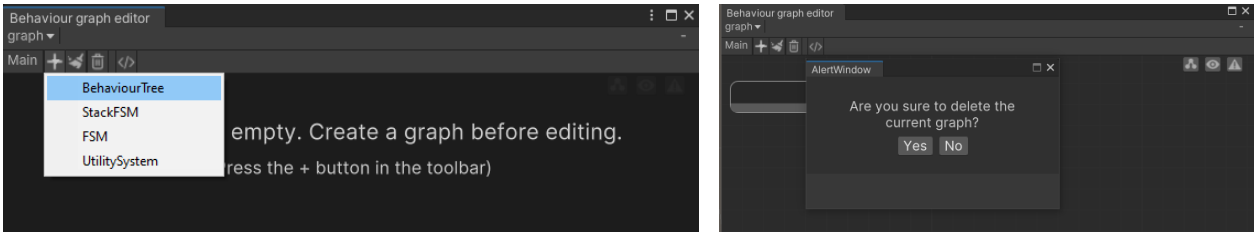
1. Menú para cambiar el grafo seleccionado.
2. Barra de herramientas del editor.
3. Editor de nodos.
4. Vista del grafo seleccionado.
5. Botones para cambiar entre el editor del grafo, de percepciones pull y de percepciones push.

4. Grafos

4.1 Añadir y borrar grafos

Para añadir un nuevo grafo al sistema seleccionado se pulsa el **botón *"+"*** de la barra de herramientas y seleccionar el tipo de grafo deseado.

Para borrar el grafo actual se pulsa el **botón con el icono de la papelera** en la barra de herramientas. Aparecerá una ventana de confirmación antes de borrarse.



9 Crear y borrar grafos

Al pulsar el botón con el icono del cepillo en la barra de herramientas se eliminarán todos los nodos del grafo.

El botón "Main" sirve para cambiar el grafo principal del sistema al grafo seleccionado actualmente.

4.2 Seleccionar y editar grafo

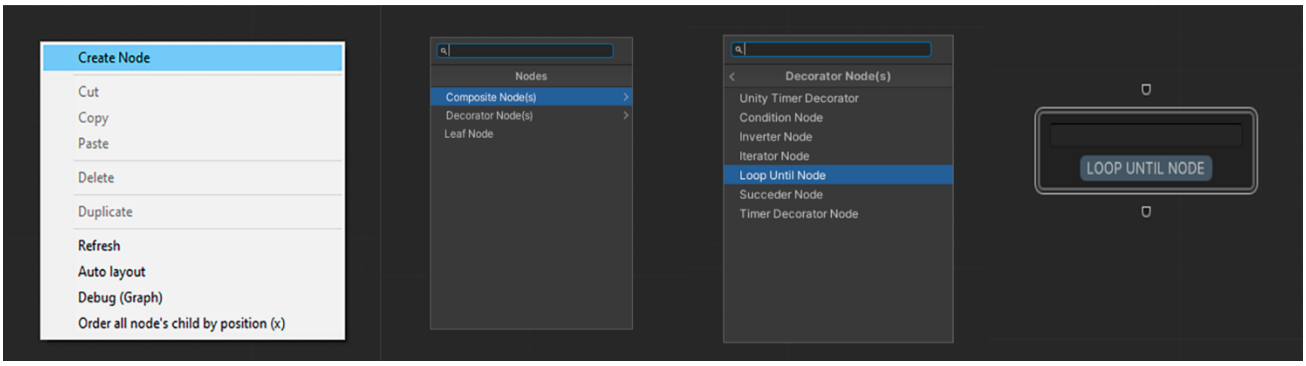
Para cambiar el grafo seleccionado se despliega el **menú de grafos** de la barra de tareas y se elige uno de los grafos del sistema. Al pulsar el **botón con el icono del grafo** de la esquina superior derecha se abre el **editor de variables del grafo**.

10 Seleccionar un grafo para editarlo

5. Nodos

5.1 Añadir y borrar nodos

Al hacer clic derecho sobre la ventana del editor se abre el menú del grafo actual. Al seleccionar la opción **"*Create Node*"** se abre otro menú que permite seleccionar un tipo de nodo de los disponibles para el grafo seleccionado. También puede abrirse directamente pulsando la tecla "*Espacio*".



11 Crear un nuevo nodo

Para borrar un nodo hay que hacer clic sobre el y pulsar la tecla suprimir.

5.2 Editar nodos

Hacer clic sobre uno de los nodos del editor abre el **editor de nodos**, que permite modificar el nombre del nodo y sus variables públicas.

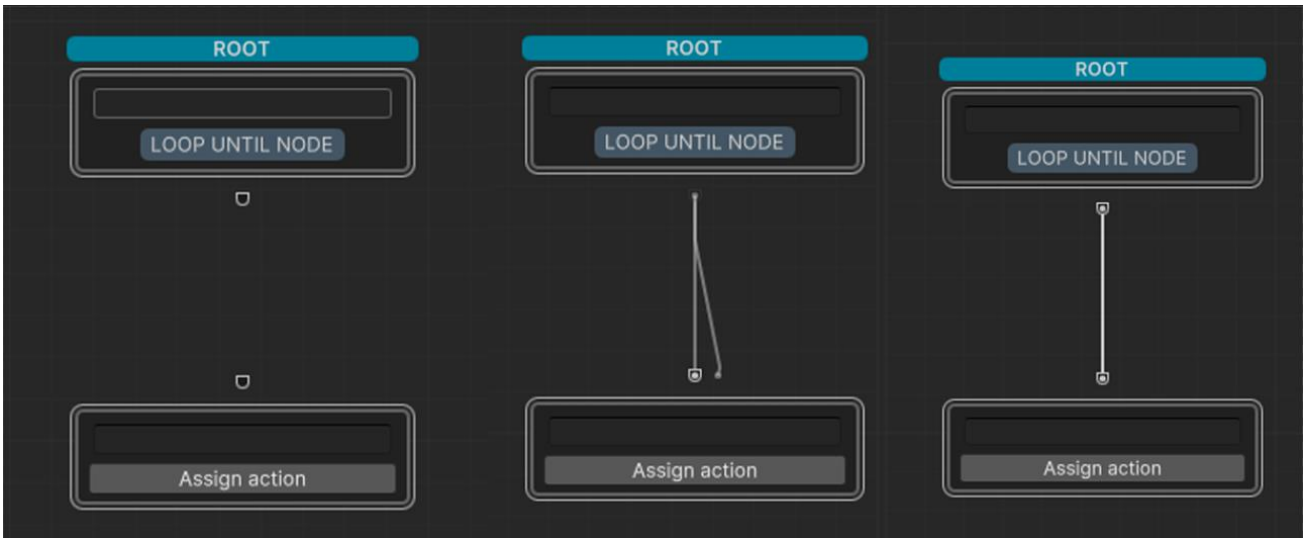
También es posible mover los nodos arrastrándolos por la interfaz.

12 Editor de nodos

5.3 Crear y eliminar conexiones entre nodos

Los nodos pueden tener uno o varios "puertos" de entrada y de salida dependiendo de su tipo. Al hacer clic sobre uno de los puertos de un nodo y arrastrar hacia un puerto del tipo contrario en otro nodo se crea una conexión entre ellos, en la que el nodo con el puerto de salida es el "padre" y el otro nodo es el "hijo".

El tipo de puerto se distingue en que los puertos de entrada tienen el lado curvo hacia dentro y los de salida hacia fuera.



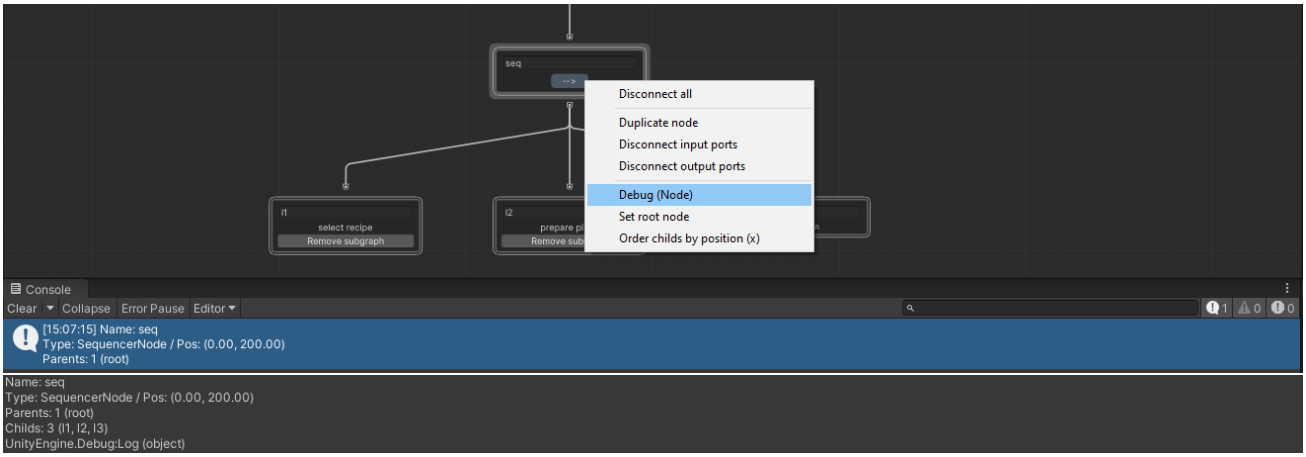
13 Crear una conexión entre dos nodos.

Para borrar una conexión hay que hacer clic sobre ella y pulsar la tecla suprimir.

5.4 Ordenar conexiones

En algunos nodos el orden de los hijos es importante y afecta a como se ejecuta el grafo. El orden por defecto será en el que se creen las conexiones, pero puede ordenarse en función de su posición (de izquierda a derecha o de arriba abajo) haciendo clic derecho sobre el nodo padre y seleccionando la opción "***order childs by position***".

Para ver el orden de conexiones de un grafo se hace clic derecho sobre él y se selecciona la opción ***"Debug node"***, que muestra por consola la información del nodo.



14 Depurar información de un nodo

6. Crear y asignar acciones

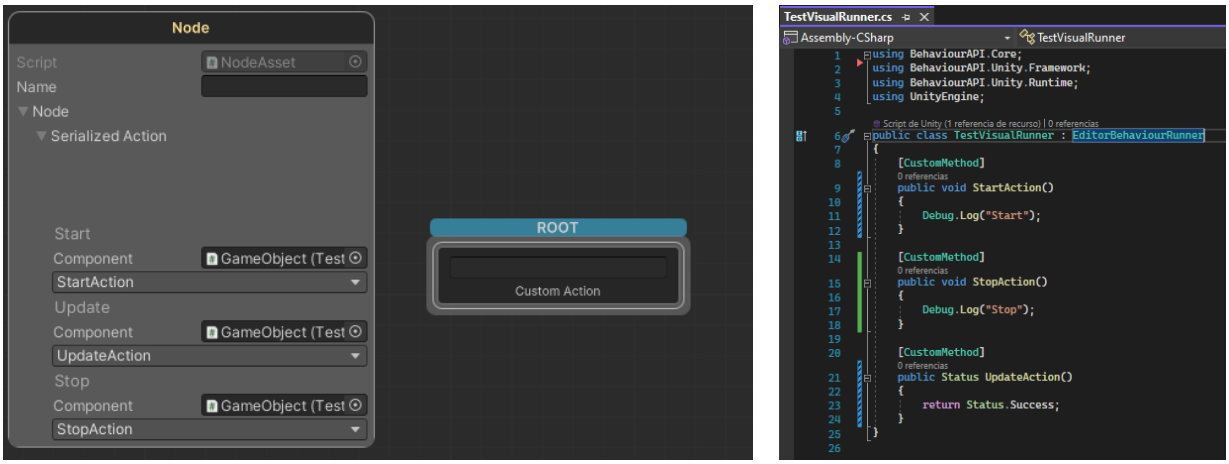
En los nodos que pueden tener acciones asignadas aparece un botón “Assign action”, que permite abrir un menú para elegir el tipo de acción. Estas acciones pueden ser de distintos tipos.

6.1 Acciones de tipo Custom Action

Permiten asignar a los eventos de la acción **métodos de componentes de unity**.

Para asignar el método se arrastra un componente a la propiedad *“Component”* y después se selecciona uno de sus métodos.

Para que un método pueda seleccionarse debe ser público, sin argumentos, su valor de retorno debe coincidir con el evento al que se asignan y deben marcarse con el atributo “CustomMethod”.



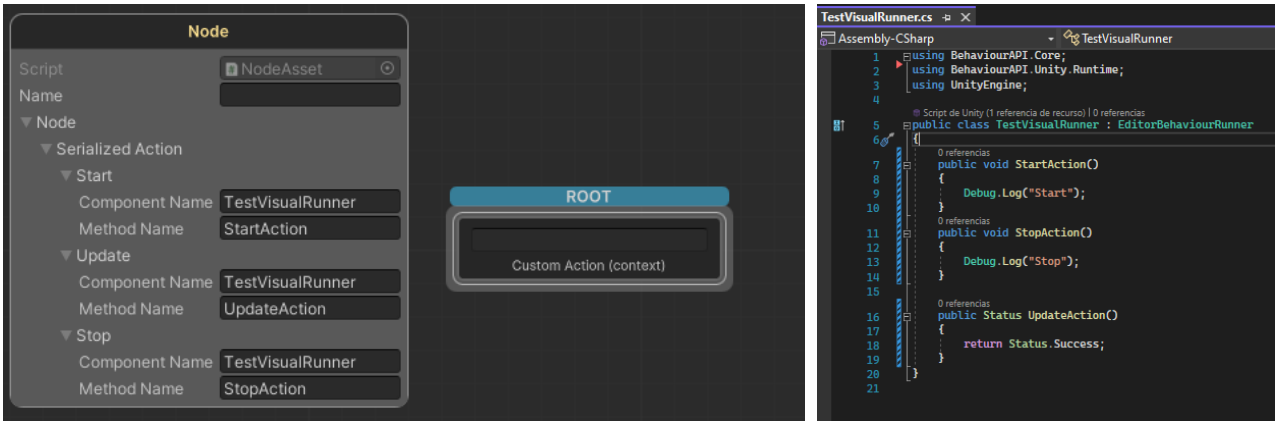
15 Crear acción de tipo CustomAction

> Este tipo de acciones no están pensadas para usarse en prefabs, porque si se usa un componente de un prefab, la acción usará el componente original en lugar del de la instancia.

6.2 Acciones de tipo Custom Context Action

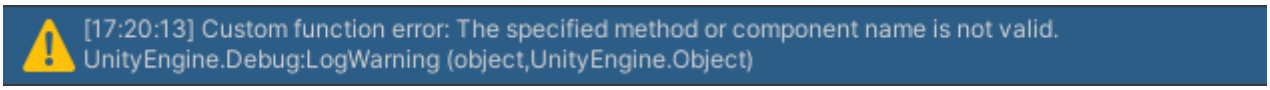
Permiten asignar a los eventos de la acción **métodos de uno de los componentes del objeto que ejecute el sistema de comportamiento**. Funcionan de forma similar al tipo anterior, pero están pensadas para usarse en prefabs, de forma que se use el componente de la instancia en lugar del componente del prefab original.

Estas acciones buscarán el componente en tiempo de ejecución, para después buscar el método especificado y asignarlo al evento correspondiente (Start/Update/Stop). El componente puede ser el mismo que ejecute el sistema (`BehaviourRunner`) o cualquier otro del objeto.



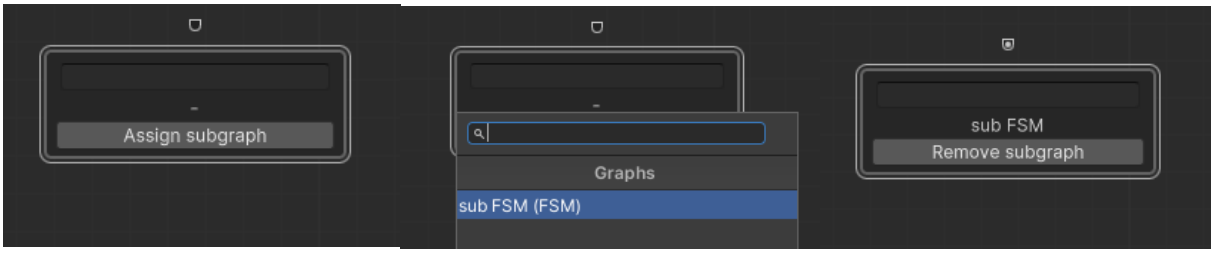
16 Crear acción de tipo CustomContextAction

Si se especifica un componente que no existe o un método no válido no se asignará y se mostrará una alerta por consola cuando se lance la aplicación.



6.3 Acciones de tipo SubgraphAction

Permiten asignar un subgrafo para que la acción lo ejecute.



17 Crear acción con subgrafo

Al pulsar el botón "*Assign Subgraph*" aparecerá un menú para escoger uno de los grafos del sistema. Si la acción ya tiene un subgrafo asignado aparecerá un botón "*Remove Subgraph*" para quitarlo.

6.4 Acciones de tipo UnityAction

Permite crear acciones usando clases personalizadas que hereden de la clase `UnityAction`. Estas acciones pueden acceder al gameobject que está ejecutando el grafo usando la variable `context` (ver contexto de ejecución de unity).

Para crear nuevas acciones de este tipo hay que seguir una serie de pasos.

1. Crear una nueva clase que herede de `UnityAction`.
2. Si se definen constructores, se debe definir también un constructor sin parámetros.
3. Se sobreescribe los métodos `Start`, `Update` y `Stop`.
4. Se puede sobreescribir el método `OnSetContext` si es necesario que la acción guarde alguna referencia de algún componente del objeto.
5. Se puede sobreescribir la propiedad *DisplayInfo* para personalizar la etiqueta que aparece en los nodos que tienen una acción de este tipo.

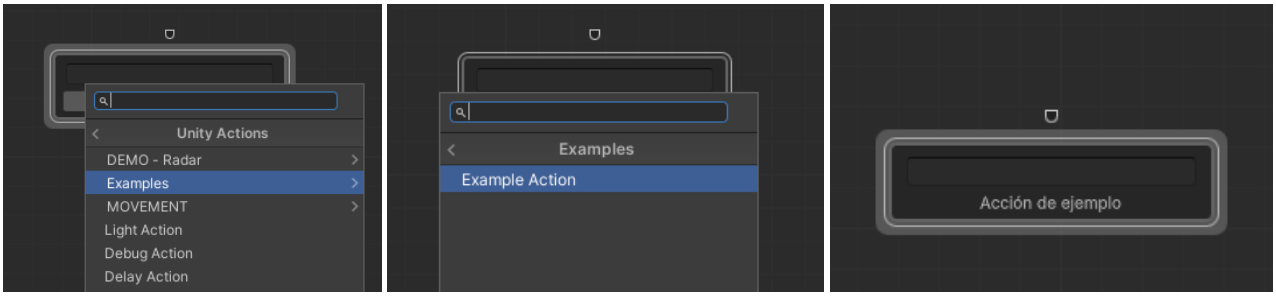
6. También se puede añadir uno o varios atributos de tipo *SelectionGroup* para incluir las acciones en grupos y facilitar el buscarlas en el menú de acciones.

```
[SelectionGroup(“Examples”)]
public class ExampleAction : UnityAction
{
    public CustomComponent customComponent;

    // Este método se lanza antes de comenzar la ejecución del grafo
    protected override void OnSetContext()
    {
        customComponent = context.GameObject.FindComponentOfType<CustomComponent>();
    }

    public override void Start { … }
    public override Status Update { … }
    public override void Stop { … }

    public override string DisplayInfo => "Acción de ejemplo";
}
```

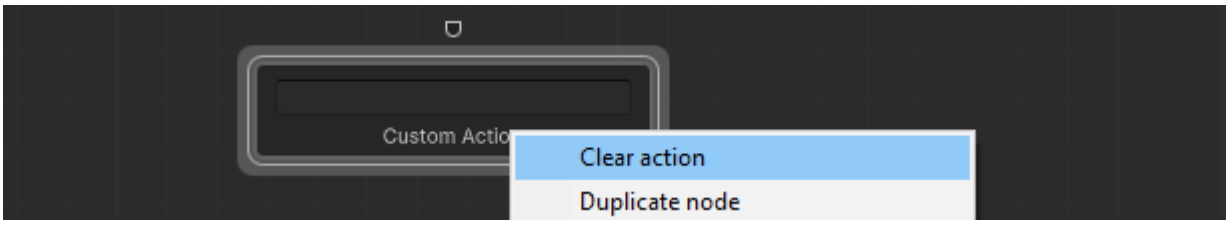


18 Crear acción de tipo UnityAction. La acción está dentro del grupo “Examples”.

El paquete ya incluye varios ejemplos de acciones en *Runtime/Extensions/Actions*.

6.5 Desasignar acciones

Para eliminar la acción de un nodo se debe hacer clic derecho sobre la etiqueta de la acción y seleccionar la opción “*Clear action*”.



19 Eliminar la acción de un nodo

7. Crear y asignar percepciones

Para crear una nueva percepción primero hay que abrir el editor de percepciones pulsando el botón con el icono del ojo en la esquina superior derecha y después pulsar el botón “Add Element” y seleccionar el tipo de percepción deseado. Debajo de este botón aparecerá la lista de percepciones creadas y al hacer clic sobre una se abrirá su editor justo debajo. Pulsar el botón “X” en un elemento de la lista borrará la percepción.

Se pueden crear varios tipos de percepciones:

7.1 Percepciones de tipo Custom Perception

Funcionan igual que las acciones de tipo `CustomAction`.

7.2 Percepciones de tipo Custom Context Perception

Funcionan igual que las acciones de tipo `CustomContextAction`.

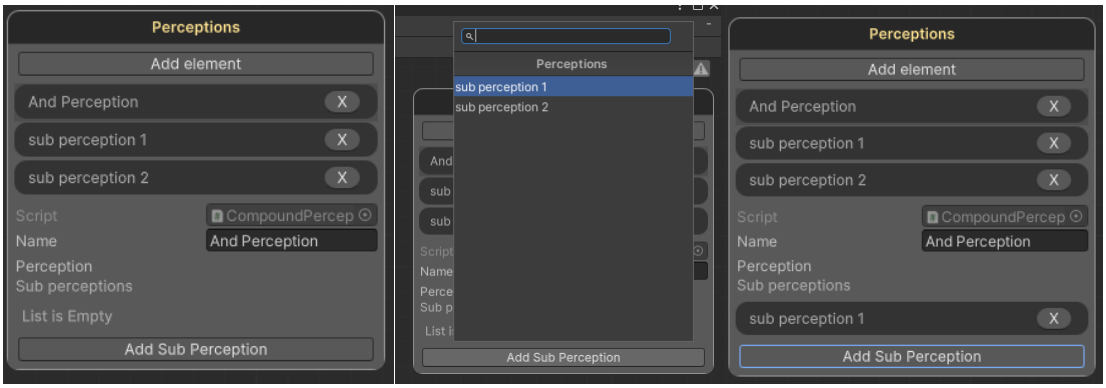
7.3 Percepciones de tipo Unity Perception

Funcionan igual que las acciones de tipo `UnityAction`. En este caso hay que extender la clase `UnityPerception` y sobreescribir los métodos `Init`, `Check` y `Update`.

El paquete ya incluye varios ejemplos de percepciones en *Runtime/Extensions/Perceptions*.

7.4 Percepciones de tipo Compound Perception

Permite crear una percepción compuesta. Al seleccionarla aparecerá un botón "*Add Subperception*" y se abrirá un menú para seleccionar una subpercepción.



20 Crear percepciones compuestas

7.5 Percepciones de tipo StatusPerception

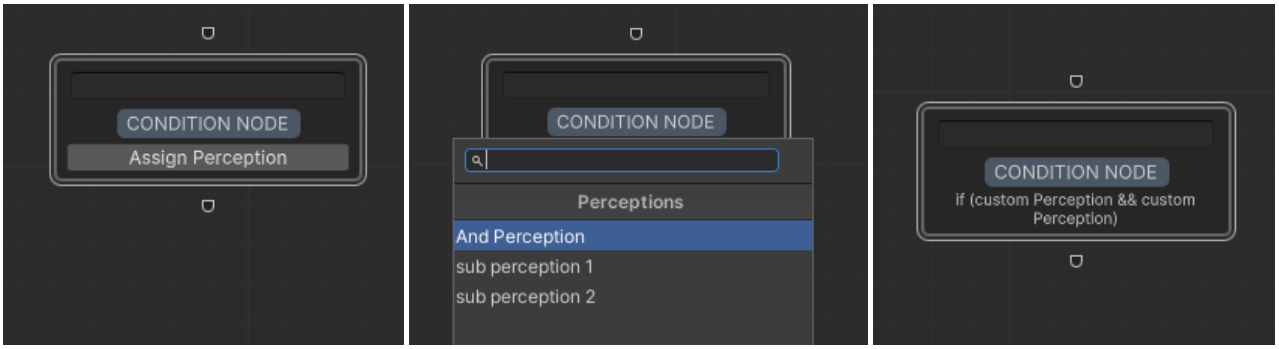
Permite crear una percepción que compruebe el estado interno de un nodo o grafo. En el editor puede asignarse el valor "`Status`" que debe tener el elemento para que se active la percepción, pero el propio elemento debe asignarse por código en el método `ModifyGraphs`.

```
IStatusHandler nodeToCheck;

protected override void ModifyGraphs()
{
   ExecutionStatusPerception perception = FindPerception<ExecutionStatusPerception>("check status");
   perception.StatusHandler = nodeToCheck;
}
```

7.6 Asignar percepción

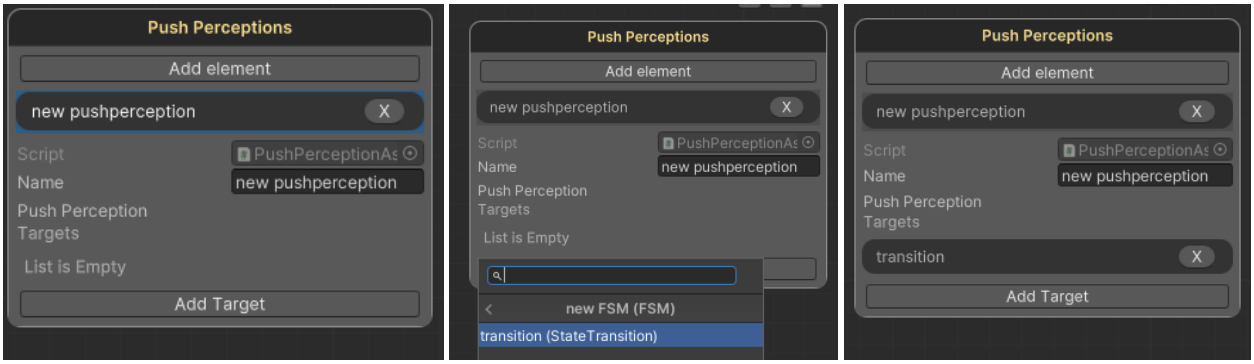
Una vez creada la percepción puede asignarse a un nodo pulsando el botón "*Assign perception*" y seleccionándola. Para quitarla se hace clic derecho sobre la etiqueta de la percepción y se selecciona la opción "*Clear perception*".



21 Asignar percepciones a un nodo

8. Crear y asignar percepciones push

Para crear una percepción push se pulsa el botón con el icono "!" de la esquina superior derecha y después se pulsa el botón "Add element". Una vez creada al hacer clic sobre una percepción push se abrirá su editor donde podemos cambiar el nombre y añadir y borrar nodos.



22 Crear percepción push

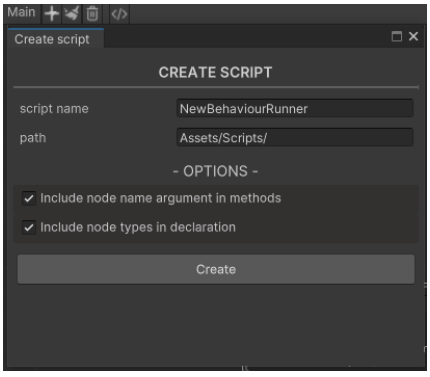
9. Generar script a partir de un sistema de comportamiento

9.1 Como generar el script

Al pulsar el botón con el icono "</>" en la barra de herramientas se abrirá una nueva ventana para generar un script a partir de los datos del sistema de comportamiento que se está editando. En esta ventana se puede especificar el nombre del script y la carpeta donde se va a guardar.

También se puede especificar si se van a usar los métodos que incluyen el nombre de los nodos o si las declaraciones de los nodos se harán con el nombre del tipo completo o con "var".

Importante: Si se especifica un nombre y una ruta que ya existe, se sobreescribirá el script.



23 Ventana para generar el script

Se recomienda que el código generado se revise siempre antes de usarlo. Es posible que el código generado tenga errores de compilación, especialmente si se incluyen elementos creados por el usuario.

9.2 Generar código de elementos creados por el usuario

Para que se pueda generar código de elementos que no incluye la API de base estos deben cumplir ciertas condiciones. Si no se cumplen el código se generará, pero no incluirá las variables de los elementos.

- En el caso de UnityActions y UnityPerceptions, para que se incluya en el código generado las variables de la acción estas deben ser públicas y los argumentos del constructor deben tener el mismo nombre (y deben tener siempre otro constructor sin parámetros).

```
public class DebugAction : UnityAction
{
    public string message;
    public DebugAction()
    {
    }
    public DebugAction(string message)
    {
        this.message = message;
    }
}
```

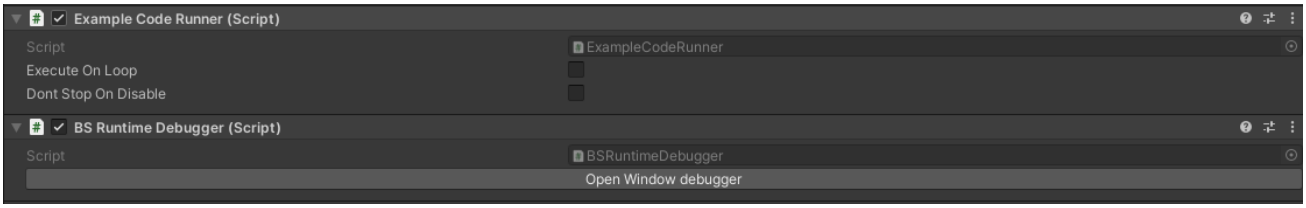
- En el caso de nodos (decoradores, factores función, etc.), para que se incluyan las variables deben ser públicas y la clase debe tener un método Setter cuyo nombre coincida con el de la variable. Es importante no declarar un constructor con parámetros.

```
public class UnityTimerDecorator : BehaviourTrees.DecoratorNode
{
    public float TotalTime;
    public UnityTimerDecorator SetTotalTime(float time)
    {
       TotalTime = time;
       return this;
    }
}
```

10. Como usar el depurador en tiempo real

10.1 Añadir el componente del depurador

Para poder ver el estado de los grafos en tiempo de ejecución hay que añadir el componente BSRuntimeDebugger al objeto que tenga cualquier script de tipo BehaviourRunner. En el inspector del script hay un botón “Open Window debugger” que si se pulsa cuando la aplicación está corriendo abre la ventana del editor en modo debug.



24 Inspector del componente del depurador

10.2 Usar el depurador con sistemas generados por código

Para que el depurador tenga acceso a los grafos generados por código debemos incluir el método `RegisterGraph` por cada uno de los grafos que se creen, pasando opcionalmente un nombre.

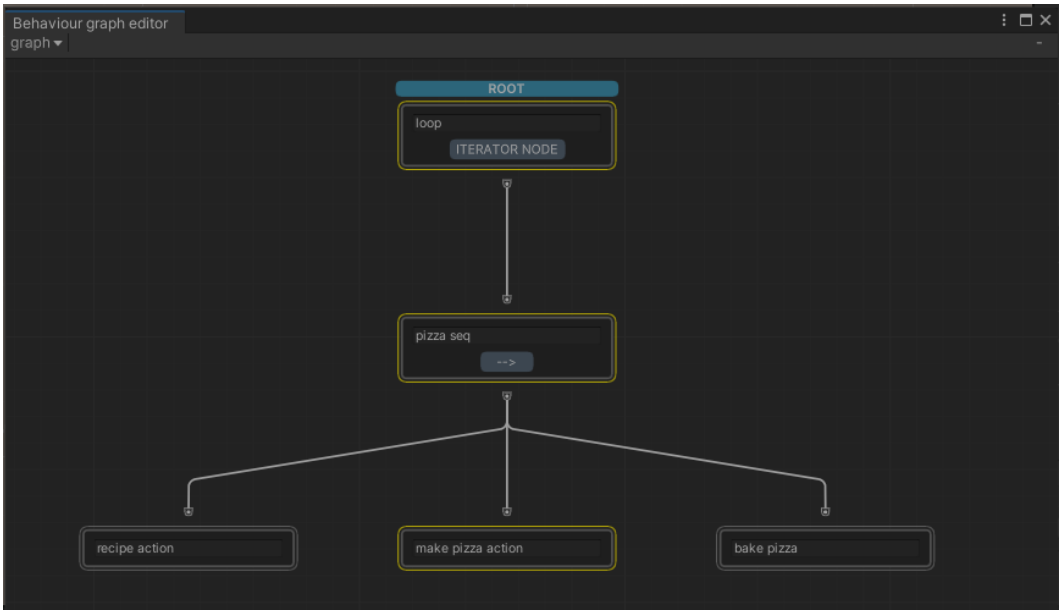
```
public class ExampleRunner : CodeBehaviourRunner
{
    protected override BehaviourGraph CreateGraph()
    {
          var fsm = new FSM();
          var subBt = new BehaviourTree();
          …
          RegisterGraph(fsm, “Main FSM”);
          RegisterGraph(subBt, “sub BT”);
          return fsm;
    }
}
```

10.3 Ventana del depurador

En este modo todas las funcionalidades de edición están desactivadas, excepto mover los nodos. Podemos cambiar el grafo que se ve con el menú de arriba a la derecha igual que en el modo de edición.

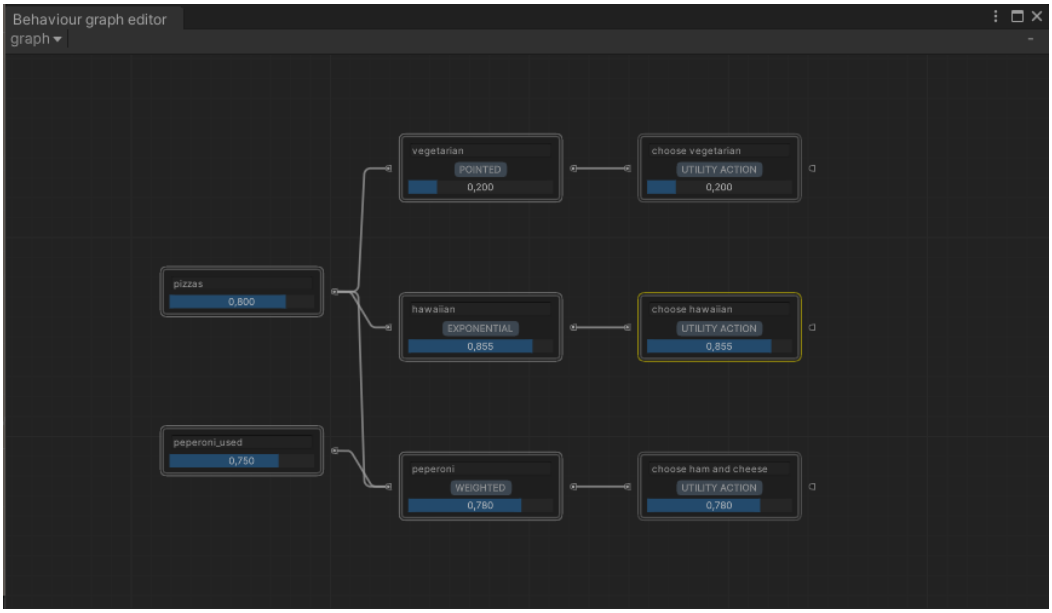
El depurador ofrece varias funcionalidades:

- Ver en tiempo real el valor “Status” de los nodos con un código de colores (gris para None, amarillo para Running, verde para Success o rojo para Failure).



25 Depurador de ejecución en un árbol de comportamiento.

- Ver en tiempo real el valor de Utilidad de los nodos de un sistema de utilidad.



26 Depurador de ejecución en un sistema de utilidad.

11.Limitaciones de la herramienta

11.1 Crear sistemas en prefabs

Cuando se crea un sistema de comportamiento usando el editor, este se guarda en objetos de tipo `ScriptableObject`, lo que genera una serie de problemas al trabajar con prefabs.

- Si un objeto que ya tiene un componente de tipo EditorBehaviourRunner con un sistema de comportamiento creado se guarda como prefab, el sistema se perderá en la instancia original del prefab, ya que unity no permite guardar referencias a elementos de escenas en assets.
- Para crear un sistema en un prefab hay que crear el prefab primero y después añadir el componente y editar el sistema desde la ventana de assets del proyecto.
- Aunque es posible, es recomendable no editar el sistema de comportamiento de una instancia de un prefab en una escena.