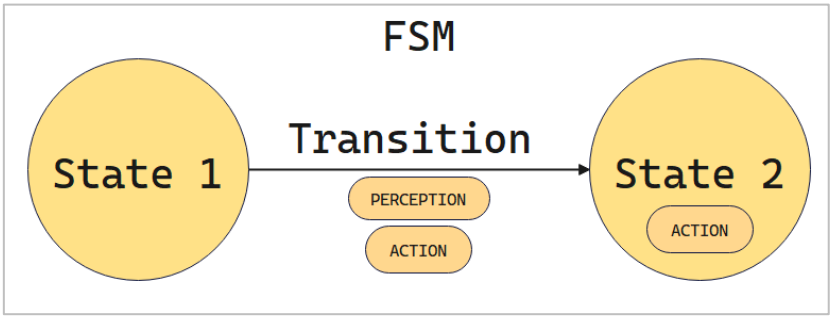
GUÍA BÁSICA DE CREACIÓN DE MÁQUINAS DE ESTADOS

Crear y ejecutar una máquina de estados

Esta guía muestra los pasos a seguir para crear la siguiente máquina de estados.



Antes de crear la máquina de estados hay que importar los espacios de nombres `BehaviourAPI.Core` y `BehaviourAPI.StateMachines`.

Para crear una máquina de estados se crea un objeto de la clase `FSM`:

```
FSM fsm = new FSM ();
```

En las máquinas de estados primero se crean los estados y después las transiciones. Después de crear todos los elementos se especifica cual es el estado inicial. Si no, será el primero en haber sido creado.

```
fsm.SetEntryState(startNode);
```

Por último, para ejecutar la máquina de estados se debe lanzar el método Start al comienzo, y después ejecutar el método Update en cada iteración.

```
FSM fsm = new FSM (); // Al principio de la ejecución se crea el grafo (p.e. Awake en Unity)
fsm.Start();  // En el primer frame se inicia la ejecución del grafo (p.e. Start en Unity)
fsm.Update(); // En cada bucle de ejecución se actualiza el grafo (p.e. Update en Unity)
```

***NOTA**: Estos métodos sólo deben ejecutarse de esta forma en el grafo principal, no en los subgrafos.*

Estados

Los estados son los elementos principales de la máquina de estados. Para crear un estado se usa el método `CreateState` y se pasa opcionalmente una acción.

Como crear acciones:

Para crear la acción hay que añadir el espacio de nombres `BehaviourAPI.Core.Actions` y crear el objeto:

```
FunctionalAction action = new FunctionalAction(StartMethod, UpdateMethod, StopMethod);
```

Los parámetros del constructor de `FunctionAction` son métodos que se ejecutarán cuando el estado comience su ejecución, en cada frame y cuando termine respectivamente. Los métodos `Start` y `Stop` deben devolver `void` y el método `Update` debe devolver `Status`.

Se pueden crear objetos de la clase FunctionalAction especificando solo algunos de los 3 métodos. Si no se especifica el método Update, la acción devolverá siempre Running y no terminará nunca.

Después de crear la acción se crea el estado (o directamente si no tiene acción). Si el estado tiene una acción asignada, su valor `Status` se actualizará con el de la acción, mientras que si no la tiene se mantendrá en `Running`.

En el ejemplo de la guía se crean dos estados, el segundo de ellos con una acción asignada.

```
State state1 = fsm.CreateState();
State state2 = fsm.CreateState(action);
```

Transiciones

Las transiciones permiten a la máquina de estados cambiar su estado actual.

Transiciones entre estados

Para crear una transición entre dos estados se usa el método `CreateTransition`, pasando como argumentos el estado origen y el estado destino. Opcionalmente se pasan también una percepción y una acción.

Como crear percepciones:

Para crear la percepción hay que añadir el espacio de nombres `BehaviourAPI.Core.Perceptions` y crear el objeto:

```
ConditionPerception perception = new ConditionalPerception(InitMethod, CheckMethod, ResetMethod);
```

Las percepciones, al igual que las acciones, se crean con tres métodos que sirven para inicializar, comprobar y resetear la percepción respectivamente. Los métodos `Init` y `Reset` deben devolver `void` y el método `Check` debe devolver un valor `bool`.

También es posible especificar solo algunos de los métodos y si el método `Check` no se especifica siempre devolverá `false`.

Existen otros tipos de percepciones:

- Percepciones compuestas: Realizan una operación lógica con el resultado de otras percepciones.

```
AndPerception and = new AndPerception(perception1, perception2, …);
OrPerception or = new OrPerception(perception1, perception2, …);
```

- Percepciones temporales: Devuelven false hasta que pasa una cantidad determinada de tiempo (en segundos).

```
TimerPerception timer = new TimerPerception(5);
```

- Percepciones de ejecución: Comprueba si el valor status de un nodo o grafo es igual a un valor. Puede servir para comunicar sistemas de comportamiento de distintos agentes.

```
Node node = …;
BehaviourGraph graph = …;
ExecutionStatusPerception stPerception = new ExecutionStatusPerception(node, StatusFlags.Running);
ExecutionStatusPerception stPerception = new ExecutionStatusPerception(graph, StatusFlags.Running);
```

Si la percepción no se especifica, la transición siempre devolverá true al comprobarse. La acción solo se ejecutará en el frame en el que se lance la transición.

En el ejemplo inicial se crean una transición del estado “State1” al estado “State2” con una percepción y una acción asignadas.

```
StateTransition transition = fsm.CreateTransition(state1, state2, perception, action);
```

Especificar cuando se comprueba una transición

Por defecto las transiciones se comprueban siempre que el estado origen sea el estado actual de la FSM, pero en algunos casos puede ser necesario que la transición solo se active cuando el estado origen haya terminado de ejecutarse. Para esto se añade un parámetro de tipo `StatusFlags` a los métodos que crean transiciones (de cualquier tipo), de forma que la transición solo se comprobará cuando el valor status del estado origen coincida con este parámetro. A continuación, se muestran algunos casos de uso de esta funcionalidad:

```
// La transición se lanzará directamente cuando la acción de s1 haya terminado:
StateTransition t = fsm.CreateTransition(s1, s2, statusFlags: StatusFlags.Finished);

// La transición se lanzará directamente cuando la acción de state1 haya terminado (con éxito):
StateTransition t = fsm.CreateTransition(s1, s2, statusFlags: StatusFlags.Success);

// La transición se lanzará directamente cuando la acción de state1 haya terminado (con fracaso):
StateTransition t = fsm.CreateTransition(s1, s2, statusFlags: StatusFlags.Failure);

// La transición sólo se comprobará cuando la acción de state1 haya terminado y se lanzará cuando se cumpla la condición de la percepción asignada.
StateTransition t = fsm.CreateTransition(s1, s2, perception, statusFlags: StatusFlags.Finished);

// La transición no se comprobará nunca (útil para hacer transiciones que solo se activen externamente con percepciones push)
StateTransition t = fsm.CreateTransition(s1, s2, statusFlags: StatusFlags.None);
```

Activar transiciones de forma externa

Para activar una transición desde fuera de la FSM se debe crear un objeto de la clase `PushPerception` y pasarle como argumento la transición o transiciones que se quieren activar.

```
PushPerception push = new PushPerception(transition1, transition2, ...);
```

Una vez creada, se puede usar el método Fire para lanzar las transiciones desde cualquier parte del código, evitando las comprobaciones en cada iteración que supondría una percepción normal.

```
push.Fire();
```

Para que una transición pueda ser activada, su estado origen debe ser el estado actual de la `FSM`.

Crear subgrafos con máquinas de estados

Crear un subgrafo dentro de una FSM

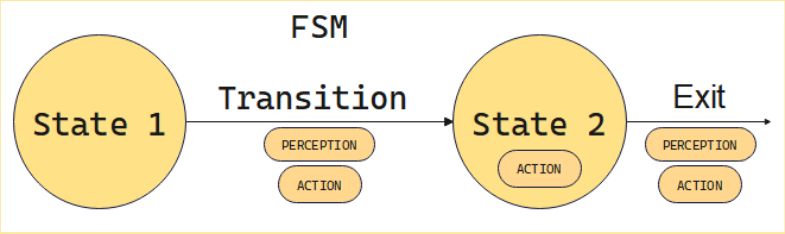
En las FSM se pueden crear subgrafos dentro de los estados usando acciones de tipo `SubsystemAction`:

```
FSM fsm = new FSM ();
BehaviourGraph subgraph = ...;
SubsystemAction action = new SubsystemAction(subgraph);
State state = fsm.CreateState(action);
```

NOTA: *Aunque también es posible, no es recomendable pasar acciones con subgrafos a transiciones, ya que estas sólo ejecutan su acción en el instante en el que son activadas.*

Salir de una submáquina de estados

Si la `FSM` es el subgrafo, para salir hay que lanzar una transición de salida.



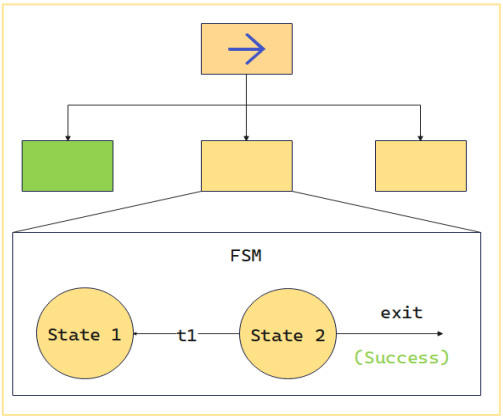
Las transiciones de salida permiten terminar la ejecución de la máquina de estados. Para crear una transición de salida se usa el método `CreateExitTransition` y se pasa como argumentos el nodo origen y el valor `Status` con el que terminará la ejecución de la `FSM` cuando la transición se active.

El valor de salida debe ser `Status.Success` o `Status.Failure`. Al igual que las transiciones entre estados, se puede pasar de forma opcional una percepción y una acción.

```
ExitTransition exit = fsm.CreateExitTransition(state2, Status.Success, perception, action);
```

El nodo que contiene la acción de submáquina actualizará su valor status al valor con el que la submáquina haya terminado su ejecución permitiendo al grafo padre continuar su ejecución.

En el ejemplo, cuando se lance la transición de salida la FSM terminará su ejecución con éxito, por lo que el nodo que la contiene también cambiará a éxito y la secuencia pasará al siguiente nodo.



Extensión: Máquinas de estados de pila

Las máquinas de estados de pila permiten almacenar en una estructura de datos de tipo pila los estados que se van recorriendo.

Para crear una fsm de pila se añade el espacio de nombres `BehaviourAPI.StateMachines.StackFSMs` y se crea un objeto de la clase `StackFSM`.

```
StackFSM stackFSM = new StackFSM();
```

Esta clase tiene los mismos métodos que una FSM normal, pero incluye dos tipos de transiciones más:

Transiciones push

Las transiciones push guardan el estado origen en la pila al lanzarse. Para crear una transición push se usa el método `CreatePushTransition`, pasando como argumentos el estado origen y el estado destino.

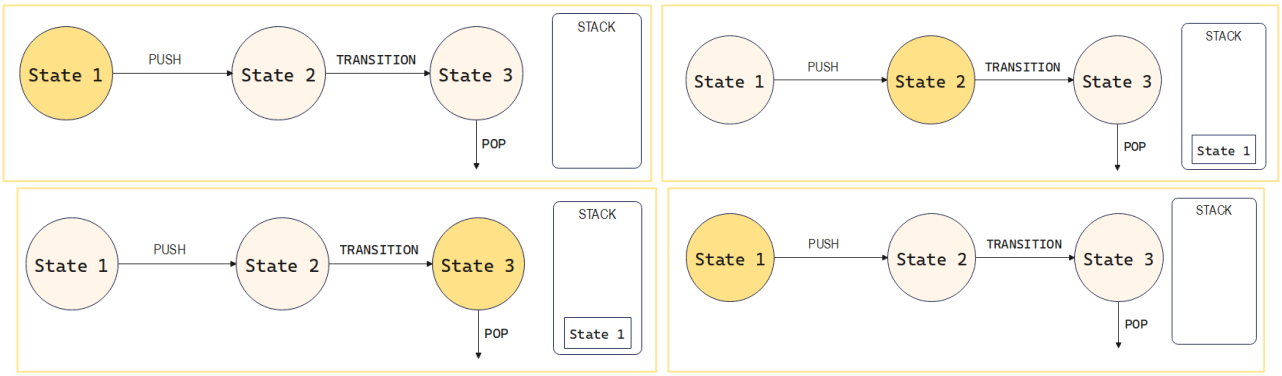
```
PushTransition push = stackFSM.CreatePushTransition(state1, state2, perception, action, ...);
```

Transiciones pop

Las transiciones pop permiten volver al último estado guardado en la pila. Para crear una transición pop se usa el método `CreatePopTransition`, pasando como argumento el estado origen.

```
PopTransition pop = stackFSM.CreatePopTransition(state1, perception, action, ...);
```

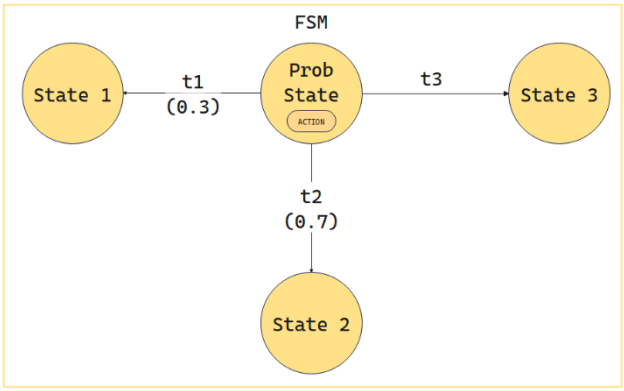
A continuación, se muestra cómo funcionan los nuevos tipos de transiciones. Cuando la FSM pasa del estado 1 al 2, pasa el estado 1 a la pila. Cuando se lanza la transición pop desde el estado 3, se saca el ultimo estado introducido a la pila y se transiciona a él.



Extensión: Estados probabilísticos

Los estados probabilísticos permiten asignar probabilidades a sus transiciones. En cada iteración, el estado elegirá aleatoriamente una de las transiciones que tengan asignada una probabilidad y solo esa será comprobada.

Para crear un estado probabilístico se usa el método `CreateProbabilisticState` pasando como parámetro opcional la acción que ejecuta el estado.



Una vez creadas las transiciones del estado, se puede especificar la probabilidad de una transición con el método `SetProbability`.
Si la probabilidad especificada es 0 o menos, se considera que esa transición no tiene probabilidad asignada y se comprobará siempre, teniendo prioridad con respecto a las transiciones que si tienen.

```
ProbabilisticState probState = fsm.CreateProbabilisticState(action);
StateTransition t1 = fsm.CreateTransition(probState, s1);
StateTransition t2 = fsm.CreateTransition(probState, s2);
StateTransition t3 = fsm.CreateTransition(probState, s3);

probState.Setprobability(t1, 0.3f);
probState.Setprobability(t2, 0.7f);
```