

# Hierarchical State Machine

## Definition

The state pattern allows objects to change behavior based on their internal state, effectively implementing finite state machines (FSM). This pattern is essential for managing AI behavior, player character states, game flow states, and animation controllers.

State machines prevent spaghetti code from proliferating conditional statements and ensure clear state transitions.

Complex AI and game flow often require hierarchical state machines (HSM) where states can contain sub-states.

Hierarchical state machines reduce code duplication by allowing states to share common behavior through parent states. When transitioning between child states, the parent state remains active and continues executing its logic.

## Implementation

In GFF, a HSM is implemented by creating a class that derives from

```
HierarchicalStateMachine<TState, TTransition>.
```

`TState` and `TTransition` are enum types. It is recommended to declare these enums inside the HSM class since they are highly specific to its implementation.

The HSM structure must be declared before updating it. It is recommended to do it in the constructor, then to call `AssertCorrectness` that verify the integrity of the HSM.

You can register a new state using the method `RegisterState`, then you register transitions to other states of the state machine with `RegisterTransitionTo`. You also can register a pass (a delegate) for any trigger type with `RegisterTransitionTo`.

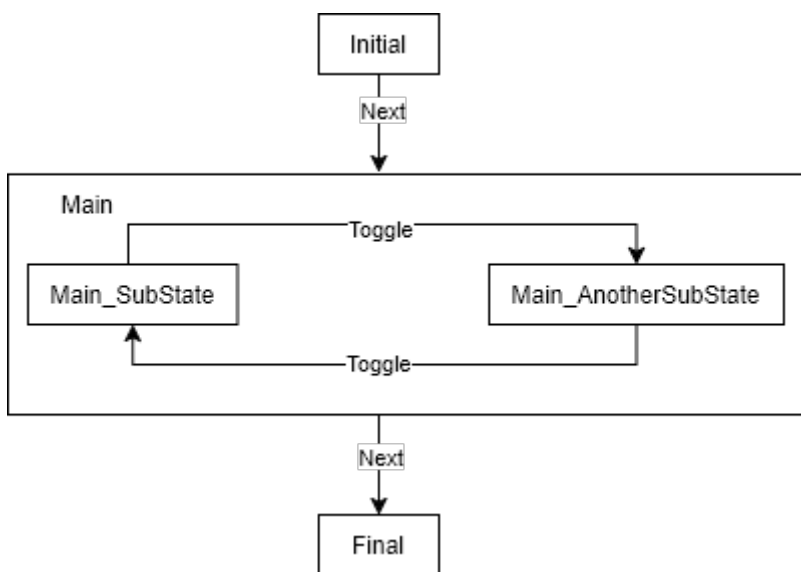
Here are the available types:

- Begin: the pass is called when entering the state.
- Update: the pass is called each update while being in the state (after Begin the first time it enters the state, and before End the last time).
- End: the pass is called before leaving the state.

Each parent state must have one and only one default child. An implicit transition is created between the parent and its default child. When a transition has a parent state as destination, the HSM will automatically transition to the default child. If you don't want to go to the default child, it means you need to declare a transition to a child state directly.

## Example

For example, if we want to create the following state machine:



```

using Core;

public partial class MyHSM : HierarchicalStateMachine<MyHSM.Id, MyHSM.Transition>
{
    public AppState() : base(Id.Bootstrap)
    {
        this.RegisterState(Id.Initial)
            .RegisterTransitionTo(Transition.Next, Id.Main);

        {
            StateInfo main =
            this.RegisterState(Id.Main)
                .RegisterPass(PassType.Begin, this.Main_Begin)

```

```

        .RegisterPass(PassType.End, this.Main_End)
        .RegisterTransitionTo(Transition.Next, Id.Final);

    outGame.RegisterState(Id.Main_SubState, defaultState: true)
        .RegisterPass(PassType.End, this.Main_SubState_Update)
        .RegisterTransitionTo(Transition.Toggle, Id.Main_AnotherSubState);

    outGame.RegisterState(Id.Main_AnotherSubState)
        .RegisterPass(PassType.End, this.Main_AnotherSubState_Update)
        .RegisterTransitionTo(Transition.Toggle, Id.Main_SubState);
}

this.RegisterState(Id.Final);

this.AssertCorrectness();
}

private void Main_Begin()
{
}

private void Main_End()
{
}

private void Main_SubState_Update()
{
}

private void Main_AnotherSubState_Update()
{
}

public enum Id
{
    Initial,

    Main,
    Main_SubState,
    Main_AnotherSubState,

```

```

        Final,
    }

    public enum Transition
    {
        Next,
        Toggle,
    }
}

```

You can also see `res://sources/application/AppState.cs` and `res://sources/gameplay/GameState.cs` as real life example. They are setup to handle the application and game control flow in GFF.

## Usage

Once your HSM is declared, you must call the Update method to run it once. During an update, the state machine will call the Update passes for all active states (from children to parent) then it will proceed to transition changes if there is pending one.

To ask for a transition change, you must call the method `void QueueTransition(TTransition transition)`. If the method is called from outside of the state machine (not in a pass during the update) the transition change is applied directly.

You can ask the state machine if you are in a specific state with the method `bool IsInState(TState state)`. It returns `true` for the current active state and all its parents. You can also ask if a specific transition is available in the current state with `bool HasTransitionAvailable(TTransition transition)`, it makes it really easy to disable actions that are not available in the game in the current state.

You can serialize the HSM. It will restore its current state on load automatically.

### Sources

<https://generalistprogrammer.com/game-design-patterns#state-pattern>

Revision #9

Created 2026-07-20 07:13:22 UTC by Adrien Allard

Updated 2026-07-26 11:24:17 UTC by Adrien Allard