Simple man: State machine Download

A state machine is a design pattern that helps to model the behavior of an object by separating it into multiple states and transitions between these states. This framework provides methods to add, remove, and switch between states.

Author: Igor-Valerii Chebotar, Kamil Siara
Email: igor.valerii.chebotar@gmail.com

Dependencies

- Simple Man - Utilities

Introduction:

State is a behavioral design pattern that lets an object alter its behavior when its internal state changes. It appears as if the object changed its class. More info about state pattern can be found here

Quick start:

1. Create Enum and declarate your states as it's values.
2. Create a state classes and implement 'IState' interface for each of them.
3. Add 'Bind' to your enum values and connect them with states classes by this way.
4. Create an empty class and inherit it from 'BaseStateMachine'.
5. Create an empty class and inherit it from 'BaseStateMachineFactory'.
6. Call 'Create' method in your factory class and receive your state machine with automatically created states.

Properties

Property name	Description
Name	The name of the state machine
States	Read-only dictionary of the states in the state machine
CurrentStateKey	The key binded with the current state of the state machine

Property name	Description
PreviousStateKey	The key binded with the previous state of the state machine
PrintLogs	A boolean property that indicates if logs should be printed

Methods

Method name	Description
AddState	Adds a state to the state machine
RemoveState	Removes a state from the state machine
SwitchState	Switches the current state to a new state
SwitchState<TState, T0>	Switches the current state to a new state with one parameter
SwitchState<TState, T0, T1>	Switches the current state to a new state with two parameters
SwitchState<TState, T0, T1, T2>	Switches the current state to a new state with three parameters
SwitchState<TState, T0, T1, T2, T3>	Switches the current state to a new state with four parameters

State class example

```
public class IdleState : IState
{
    //Called on state machine enters to this state
    public void Start()
    {

    }

    //Called on state machine exits from this state
    public void Stop()
    {

    }
}
```

States binding example

```
//Use 'Bind' attribute to bind Enum key and state class type
public enum EStates
{
    None = 0,

    [Bind(typeof(IdleState))]
    Idle = 1,

    [Bind(typeof(FloatingState))]
    Floating = 2,

    [Bind(typeof(JumpingState))]
    Jumping = 3
}
```

Your state machine class

```
//Inherit your state machine from the base state machine class.
public class ExampleStateMachine : BaseStateMachine<EStates>
{
    //You can give a special name for your state machine if you need (optional).
    //You can also add a custom debug logger as a second parameter (optional).
    public ExampleStateMachine() : base("My state machine")
    {

    }
}
```