**Working with State Machines in the .NET Framework**

In this article I will introduce the readers to the state machine model and also create a simple working state machine sample using the .NET framework in C# language.

**What is a State Machine?**

Any device that changes its state from one to another due to some actions are defined to be state machines. For example an ATM machine, traffic signal, remote control, the computer itself, etc. Most software applications also depend and operate based on the state. There are two types of state machines.

1. Finite state machine – A state machine holding a defined set of states and they work within that.

2. Infinite state machine – Here the states can be many and cannot be predefined.

Most common state machines are finite ones.

**Composition of a State Machine**

The following things collectively make an effective finite state machine.

**State:**

A defined set of states. At any point of time the state machine will be in any one of the defined states. For example Red, Green and Yellow in a traffic signal system.

**State Transition:**

A state machine changing its state from one to another is called State Transition. Usually there will be a state transition table built that will contain the state sequences and also metadata saying which event will cause which state transition.

**Triggers:**

Triggers are the points that induce the state transition in a state machine.

**Events or Actions:**

Actions are performed by the state machines when a particular state is attained and exited. There will be Entry and Exit actions for each state.

**Guard Condition:**

This is the piece of component that validates the transition and ensures that no invalid state changes are carried out.

**State Machine Vs State Pattern**

State pattern is one among the Gang of Four patterns. Most people have an impression that state machine and state pattern are different but the truth is state pattern is one way to implement a state machine. There are other ways available to build sophisticated state machines including all the components mentioned in the above section.

**Traffic Signal State Machine Example**

Let us create a simple state machine using a state pattern in .NET framework. The state machine that we are going to build is a Traffic Signaling system, which changes the signals within specific time intervals once the system is started.

Create a state interface named IState, which will be implemented by all the available states.

1. namespace FiniteStateMachine
2. {
3. public interface ISignalState
4. {
5. void Enter(TrafficSystem system);
6. }
7. }

Next let us create the states for the traffic signal. The states will be Red, Green and Yellow.

1. namespace FiniteStateMachine
2. {
3. public class RedState : ISignalState
4. {
5. const int SIGNAL\_TIME = 5000;
7. public void Enter(TrafficSystem system)
8. {
9. Console.ForegroundColor = ConsoleColor.Red;
10. Console.WriteLine("Entered Red.");
11. Thread.Sleep(SIGNAL\_TIME);
12. system.NextTrafficSignal = new GreenState();
13. system.ChangeSignal();
14. }
15. }
16. }
17. namespace FiniteStateMachine
18. {
19. public class GreenState : ISignalState
20. {
21. const int SIGNAL\_TIME = 10000;
23. public void Enter(TrafficSystem system)
24. {
25. Console.ForegroundColor = ConsoleColor.Green;
26. Console.WriteLine("Entered Green.");
27. Thread.Sleep(SIGNAL\_TIME);
28. system.NextTrafficSignal = new YellowState();
29. system.ChangeSignal();
30. }
31. }
32. }
33. namespace FiniteStateMachine
34. {
35. public class YellowState : ISignalState
36. {
37. const int SIGNAL\_TIME = 3000;
39. public void Enter(TrafficSystem system)
40. {
41. Console.ForegroundColor = ConsoleColor.Yellow;
42. Console.WriteLine("Entered Yellow.");
43. Thread.Sleep(SIGNAL\_TIME);
44. system.NextTrafficSignal = new RedState();
45. system.ChangeSignal();
46. }
47. }
48. }

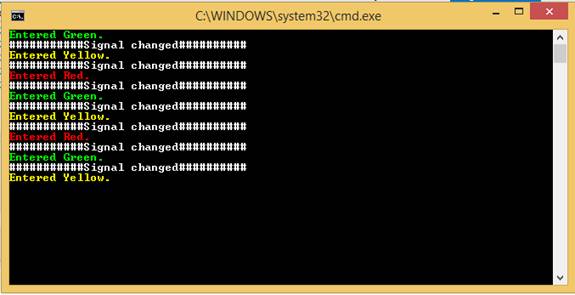
Now comes the traffic system that holds an initial state, which is Green and then changes its context state periodically. It exposes a Start function, which is a handle for the clients to kick start the traffic system.

1. namespace FiniteStateMachine
2. {
3. public class TrafficSystem
4. {
5. public ISignalState NextTrafficSignal { get; set; }
6. public void Start()
7. {
8. NextTrafficSignal = new GreenState();
9. NextTrafficSignal.Enter(this);
10. }
12. public void ChangeSignal()
13. {
14. Console.ForegroundColor = ConsoleColor.White;
15. Console.WriteLine("###########Signal changed##########");
16. NextTrafficSignal.Enter(this);
17. }
18. }
19. }

Now we are good to hook it up to the client, which is a console app Main function in our case.

1. class Program
2. {
3. static void Main(string[] args)
4. {
5. TrafficSystem system = new TrafficSystem();
6. system.Start();
8. Console.ReadLine();
9. }
10. }

Run the application and you should see the signal changing behavior as shown in Fig 1.0.

  
Fig 1.0: Signal Changing Behavior

It is not only about creating the state machine application but also can be extended into creating state machine frameworks that can be used by other applications. .NET framework has integrated the state machine framework in the form of Workflow foundation.

Happy reading!

