Иерархический конечный автомат. Работа светофора

В работе мы построим конечный автомат, представляющий светофор. Конечный автомат светофора будет иметь включенное и выключенное состояние, указывающее, работает он или нет. Кроме того, когда он находится во включенном состоянии, у него будет три подсостояния: красное, желтое и зеленое, указывающие, какой из индикаторов горит в данный момент:

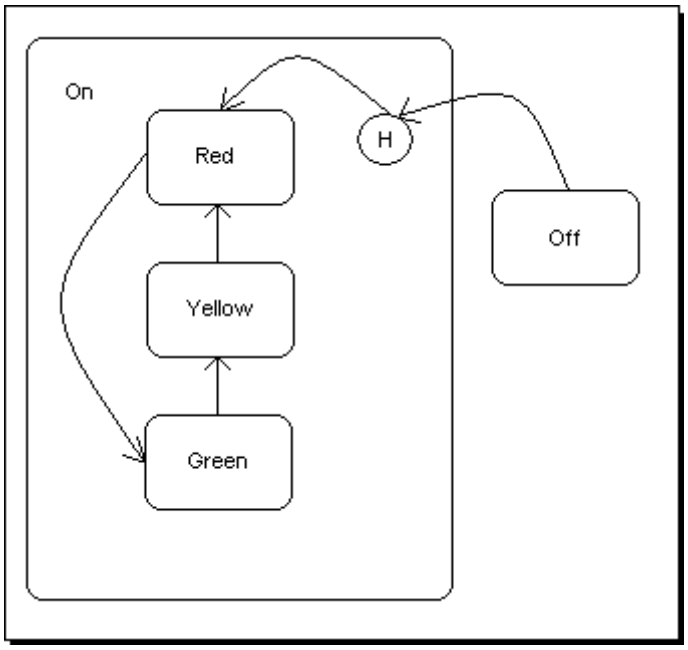

Рисунок 1. Диаграмма состояний конечного автомата светофора.

В состоянии *On* конечный автомат будет периодически менять, какой из индикаторов горит (с красного на зеленый, желтый и обратно на красный). Конечно, реальный светофор был бы более сложным, но этой упрощенной версии будет достаточно для примера реализации иерархического состояния.

Реализация иерархических конечных автоматов

Давайте перейдем к коду. Вот начало TrafficLightкласса конечного автомата

```
использование System;
использование Sanford.Многопоточность;
использование Sanford.StateMachineToolkit;

Пространство имен Traffic Lightdemo
{
 открытый  класс Traffic Light: ActiveStateMachine
 {
 публичное  перечисление EventID
 {
```

```
 Утилизировать,
 Включение,
выключение,
время истекло
 }

 общедоступный  StateID перечисления
 {
 Вкл.,
Выкл.,
красный,
желтый,
 Зеленый,
удаляемый
 }

 частное состояние включено, выключено, красный, желтый,
зеленый, удалено;

 частный планировщик DelegateScheduler = новый планировщик
DelegateScheduler();

 общественный светофор ()
 {
on = новое состояние((int)StateID.On, новый обработчик записи
(EntryOn));
 выкл = новое состояние ((int)StateID.Выкл., новый
EntryHandler(EntryOff));
 красный = новое состояние((int)StateID.Red, новый
EntryHandler(EntryRed));
 желтый = новое состояние ((int)StateID.Желтый,
новый EntryHandler(EntryYellow));
 зеленый = новое состояние ((int)StateID.Зеленый,
новый EntryHandler(EntryGreen));
 disposed = новое состояние ((int)StateID.Disposed,
новый EntryHandler(EntryDisposed));

 на.подсостояниях.Добавить (красный);
на.Substates.Добавить (желтый);
на.Substates.Добавить (зеленый);

 on.initialState = красный;

 вкл.Тип истории = тип истории.Мелкий;

 Инициализировать (выкл.);
```

```
}

#Методы входа / выхода из региона
        #общедоступное
          переопределение  конечного региона void Dispose()
{
#защита региона
           , если(IsDisposed)
{
Возврат;
}

#конечный регион
Отправить((int)EventID.Dispose);
}

частный  делегат  аннулирует SendTimerDelegate();

частный  void SendTimerEvent()
{
Отправить((int)EventID.Время истекло);
}
}
}
```

Начальное состояние

Суперсостояния имеют начальное состояние. Начальное состояние сверхсостояния является одним из его подсостояний и является состоянием, которое вводится, когда сверхсостояние является целью перехода состояния. С практической точки зрения это означает, что для нашего конечного автомата светофора это означает, что, поскольку включенное состояние является суперсостоянием для красного, желтого и зеленого подсостояний, ему требуется начальное состояние, и это начальное состояние будет одним из его подсостояний. Сделаем красное подсостояние начальным состоянием для сверхсостояния *On*. При первом вводе состояния *On* оно, в свою очередь, перейдет в состояние *Red*. Реализация этого - просто вопрос присвоения подсостояния свойству супергосударства InitialState. В этом случае свойство *red* Stateприсваивается свойству on State.InitialState. Необходимо соблюдать осторожность, потому что, если Stateon еще не был переведен в подсостояние для сверхсостояния, будет выдано исключение, когда оно будет переведено в начальное состояние сверхсостояния.

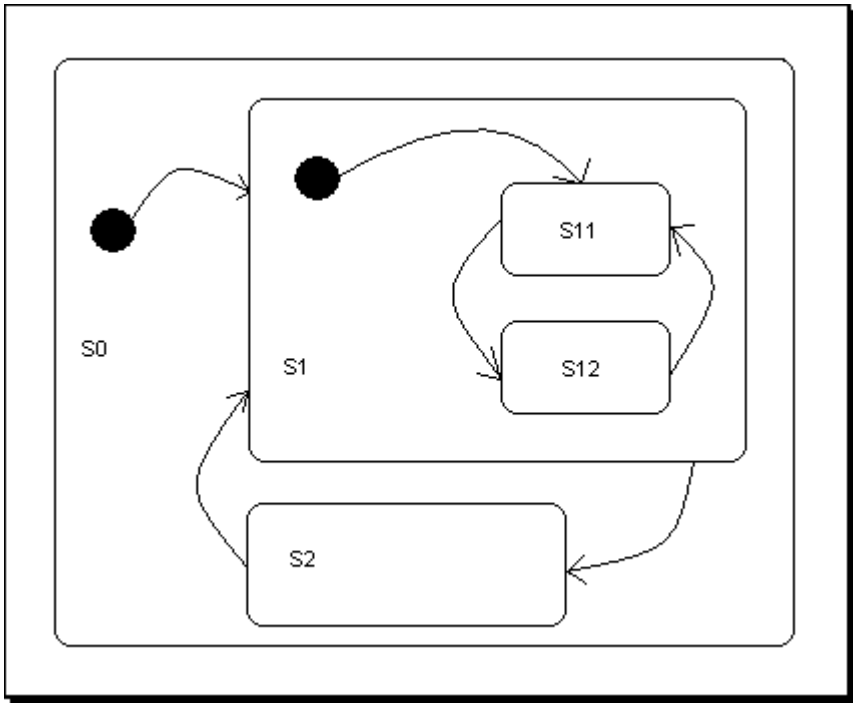

Рисунок 2. Иерархический конечный автомат

На рисунке 2 показан иерархический конечный автомат. Подсостояния вложены в свои суперсостояния. Сплошной круг, соединенный линией и заканчивающийся стрелкой, указывает на начальное состояние.

Тип истории

Обратите внимание в TrafficLightкоде, что *On* для свойства 's HistoryType установлено значение Shallow. Суперсостояния могут иметь тип истории. Тип истории определяет, насколько глубоко супергосударство будет помнить, какое из его подсостояний было активным при выходе. Существует три типа истории, None, Shallow, и Deep. С типом истории None сверхсостояние не будет помнить, какое подсостояние было последним активным, и всегда будет нацеливаться на его начальное состояние при его вводе.

Тип Shallow истории указывает, что сверхсостояние будет помнить, какое подсостояние было активным на один уровень ниже. Может быть более одного уровня вложенности с суперсостояниями и подсостояниями (см. рисунок 2). Тип Shallowистории переходит только на один уровень вниз, после чего подсостояния будут нацелены на свои начальные состояния при повторном входе.

Тип Deep истории указывает, что сверхсостояние будет помнить, какое подсостояние было активным, вплоть до нижней части иерархии состояний. Когда

вводится суперсостояние, оно, в свою очередь, переходит в последнее активное подсостояние, которое, в свою очередь, переходит в свое последнее активное подсостояние и так далее. В случае, когда существует только один уровень подсостояний, как в случае с our TrafficLight, Shallowи Deep типы истории эквивалентны.

Тип Shallow истории для on Statefor our TrafficLightна практике означает, что он будет помнить, какой индикатор был включен, когда он выключен и снова включен. Таким образом, если зеленый свет был включен, когда светофор был выключен, он запомнит, когда его снова включат, и зажжет зеленый свет (укажите зеленое состояние).

Переходы

Если посмотреть на TrafficLight код, то можно заметить, что переходы еще не добавлены. Давайте более подробно опишем, как TrafficLight должен вести себя:

Когда TrafficLighton находится в выключенном состоянии, он переходит в состояние *On* при получении TurnOn события. И когда TrafficLighton находится во включенном состоянии, он переходит в выключенное состояние при получении TurnOff события, независимо от того, в каком подсостоянии он находится. Находясь в одном из подсостояний *On*, конечный автомат ответит на TimerElapsed событие. Когда он получает TimerElapsed событие, находясь в одном из подсостояний *On*, он переходит в другое состояние в зависимости от того, в каком подсостоянии он находится в данный момент. Например, если он находится в красном подсостоянии, он перейдет в Зеленое подсостояние.

Когда вводится одно из подсостояний *On*, он будет использовать DelegateScheduler объект для планирования следующего события таймера. DelegateScheduler Класс принадлежит Sanford.Threading пространству имен.

Мы хотим, чтобы конечный автомат оставался в Красном и Зеленом подсостояниях дольше, чем в Желтом подсостоянии. Таким образом, при входе в красное или зеленое подсостояние событие таймера будет длиться дольше, чем при входе в Желтое подсостояние. Это имитирует поведение реального светофора.

```
общественный светофор ()
{
```

```
 on = новое состояние((int)StateID.On, новый
EntryHandler(EntryOn));
 выкл = новое состояние ((int)StateID.Выкл., новый
EntryHandler(EntryOff));
 красный = новое состояние((int)StateID.Red, новый
EntryHandler(EntryRed));
 желтый = новое состояние ((int)StateID.Желтый, новый
EntryHandler(EntryYellow));
 зеленый = новое состояние ((int)StateID.Зеленый, новый
EntryHandler(EntryGreen));
 disposed = новое состояние ((int)StateID.Disposed,
новый EntryHandler(EntryDisposed));

 на.подсостояниях.Добавить (красный);
на.Substates.Добавить (желтый);
на.Substates.Добавить (зеленый);

 on.initialState = красный;

 вкл.Тип истории = тип истории.Мелкий;

 Переход trans = новый переход (выключен);
включен.Переходы.Добавьте((int)EventID.Отключение, переход);

 trans = новый переход (включен);
выключен.Переходы.Добавьте((int)EventID.TurnOn, trans);

 trans = новый переход (зеленый);
 красный.Переходы.Добавьте((int)EventID.Время истекло, trans);

 trans = новый переход (желтый);
 Зеленый.Переходы.Добавьте((int)EventID.Время истекло, trans);

 trans = новый переход (красный);
 Желтый.Переходы.Добавьте((int)EventID.Время истекло, trans);

 trans = новый переход (удален);
выключен.Переходы.Добавить((int)EventID.Dispose, trans);
 trans = новый переход (удален);
 вкл.Переходы.Добавить((int)EventID.Dispose, trans);

 Инициализировать (выкл.);
}
```

И вот методы ввода для состояний. Это методы, которые будут вызываться при вводе их состояний:

```
методы ввода / вывода частного  региона # EntryOn()
{
 планировщик.Запустить();
}

частный  void EntryOff()
{
 планировщик.Stop();
 планировщик.Очистить ();
}

private void EntryRed()
{
 планировщик.Добавить(1, 5000, новый
SendTimerDelegate(SendTimerEvent));
}

private void EntryYellow()
{
 планировщик.Добавить(1, 2000, новый
SendTimerDelegate(SendTimerEvent));
}

приватная  пустая запись Green()
{
 планировщик.Добавить(1, 5000, новый
SendTimerDelegate(SendTimerEvent));
}

private void EntryDisposed()
{
 планировщик.Dispose();

 Dispose(истина);
}
```

Обратите внимание на метод ввода для состояния *On*. Он запускает DelegateScheduler. Помните, что состояние *On* является сверхсостоянием. Когда он вводится, он, в свою очередь, входит в одно из своих подсостояний. Какое подсостояние он вводит, зависит от текущего параметра истории. Если состояние *On* вводится в

первый раз, оно перейдет в исходное состояние. Таким образом, после вызова метода ввода состояния *On* конечный автомат, в свою очередь, вызовет один из методов ввода состояния состояния *On*. Например, если следующим будет введено состояние Red, оно вызовет метод ввода для подсостояния Red. Если вы посмотрите на этот метод, вы увидите, что при вводе красного подсостояния он планирует событие синхронизации с DelegateScheduler объектом. По истечении этого временного события DelegateScheduler будет вызван делегат, переданный. Здесь мы передаем ему делегат, представляющий метод, который просто отправляет событие на конечный автомат, сообщая ему, что событие таймера истекло. Также обратите внимание, что при входе в выключенное состояние конечные машины останавливают DelegateScheduler. В выключенном состоянии нет причин для его запуска. Внешний мир может определить, когда TrafficLight состояние изменилось, прослушивая TransitionCompleted событие.

Вот TrafficLight конечный автомат во всей его полноте:

```
использование System;
использование Sanford.Многопоточность;
использование Sanford.StateMachineToolkit;

Пространство имен Traffic Lightdemo
{
 Светофор публичного  класса: StateMachine
 {
 публичное  перечисление EventID
 {
 Утилизировать,
 Включение,
выключение,
время истекло
 }

 общедоступный  StateID перечисления
 {
 Вкл.,
Выкл.,
красный,
желтый,
 Зеленый,
удаляемый
 }
```

```
 частное состояние включено, выключено, красный, желтый,
зеленый, удалено;

 частный планировщик DelegateScheduler = новый планировщик
DelegateScheduler();

 общественный светофор ()
 {
on = новое состояние((int)StateID.On, новый обработчик записи
(EntryOn));
 выкл = новое состояние ((int)StateID.Выкл., новый
EntryHandler(EntryOff));
 красный = новое состояние((int)StateID.Red, новый
EntryHandler(EntryRed));
 желтый = новое состояние ((int)StateID.Желтый,
новый EntryHandler(EntryYellow));
 зеленый = новое состояние ((int)StateID.Зеленый,
новый EntryHandler(EntryGreen));
 disposed = новое состояние ((int)StateID.Disposed,
новый EntryHandler(EntryDisposed));

 на.подсостояниях.Добавить (красный);
на.Substates.Добавить (желтый);
на.Substates.Добавить (зеленый);

 on.initialState = красный;

 вкл.Тип истории = тип истории.Мелкий;

 Переход trans = новый переход (выключен);
включен.Переходы.Добавьте((int)EventID.Отключение, переход);

 trans = новый переход (включен);
выключен.Переходы.Добавьте((int)EventID.TurnOn, trans);

 trans = новый переход (зеленый);
 красный.Переходы.Добавьте((int)EventID.Время истекло, trans);

 trans = новый переход (желтый);
 Зеленый.Переходы.Добавьте((int)EventID.Время истекло, trans);

 trans = новый переход (красный);
 Желтый.Переходы.Добавьте((int)EventID.Время истекло, trans);

 trans = новый переход (удален);
выключен.Переходы.Добавить((int)EventID.Dispose, trans);
```

```
 trans = новый переход (удален);
 вкл.Переходы.Добавить((int)EventID.Dispose, trans);

 Инициализировать (выкл.);
 }

 #Методы ввода / выхода из региона
         private  void EntryOn()
 {
планировщик.Запустить();
 }

 частная  пустота EntryOff()
 {
планировщик.Stop();
 планировщик.Очистить ();
 }

 private void EntryRed()
 {
планировщик.Добавить(1, 5000, новый
SendTimerDelegate(SendTimerEvent));
 }

 private void EntryYellow()
 {
планировщик.Добавить(1, 2000, новый
SendTimerDelegate(SendTimerEvent));
 }

 приватная  пустая запись Green()
 {
планировщик.Добавить(1, 5000, новый
SendTimerDelegate(SendTimerEvent));
 }

 private void EntryDisposed()
 {
планировщик.Dispose();

 Dispose(истина);
 }

 #endregion
         публичное  переопределение  void Dispose()
 {
```

```
 #защита региона
            , если(IsDisposed)
 {
 Возврат;
 }

 #конечный регион
 Отправить((int)EventID.Dispose);
 }

 частный  делегат  аннулирует SendTimerDelegate();

 частный  void SendTimerEvent()
 {
 Отправить((int)EventID.Время истекло);
 }
 }
}
```