CS7056 – Lab 1 – Finite State Machines

This project is based on the WestWorld environment described in Buckland, chapter 2. Download and install the FSM Kit source code from the course home page: https://www.cs.tcd.ie/Mads.Haahr/CS7056/

The kit includes code for implementing FSMs and some sample agents based on the West World scenario from Buckland's book, but it's in C# rather than C++. We will be extending this code base during the course.

Task 1: Basic FSM in C#

See description in Buckland, pp.48-50.

Use the supplied code base Scripts.zip to get started. Create a new unity 2D project, drop the Scripts folder into the Assets folder, create a new empty game object, and drag & drop the "Bob" script onto the object. Run the project and Bob should start doing his thing.

To see the output from the code running, you need to make sure console window is open.

Have a look at the code and try to get an understanding of how it works. It corresponds quite closely to the structure used in Buckland. Each Agent object has its own StateMachine object that implements its behaviour. The FSM Kit also has support for state blips and messaging between agents, as described in Buckland.

State and Agent are both implemented as abstract base classes in C#.

The State class has a method Execute() that takes an Agent object as a parameter and switches state for that agent.

The Agent class has a method Update() that invokes Execute() for the agent's current state and a ChangeState() method that allows that state object to change the agent's state.

You will want to implement a couple of sample agents too (Bob and Elsa) but you can do that at any point during the lab that you feel is suitable.

Task 2: Generics

See Buckland, p.62. Turn your FSM implementation into a C# generic to allow better reusability.

Unity tutorial:

http://unity3d.com/learn/tutorials/modules/intermediate/scripting/generics

Task 3: The StateMachine Class

See Buckland, pp.64-65.

Fix up your design such that most of the code from the Agent class gets moved to a new class StateMachine that encapsulates all the code that has to do with state machines. The idea is that each agent object will own its own StateMachine object. The only method you should retain in Agent should be one method Update() that invokes the Update() method for that agent's StateMachine object.

Task 4: Add New Locations

Now add the following new locations to the West World code base:

- OutlawCamp
- SheriffsOffice
- Undertakers
- Cemetery

Task 5: Add an Outlaw

Design and implement an Outlaw agent for West World. First, design the agent's behaviour using an FSM diagram, then code it. You can look at the state machine diagrams from Buckland and the code for the Miner and MinersWife agents but please don't copy the code directly – make sure you understand what it does. Make sure your Outlaw gets some interesting behaviour, such as the following:

• The Outlaw agent should start in the OutlawCamp location.

• The Outlaw agent could lurk in the OutlawCamp and Cemetery locations for a random number of cycles and occasionally go and rob the Bank.

• When the Outlaw agent robs the Bank, he gets a random amount of gold, e.g., in the [1,10] range. (*Random.Range(1,11)*)

Now create an Outlaw agent (e.g., called Jesse) and add him to West World.

Task 6: State Blips

See Buckland, p.63. Add code to your StateMachine class to support global state and state blips.

Task 7: Messaging

See Buckland, pp.67-82. Add messaging capabilities to your FSM implementation. Messages are handled with delegate and event pairs. An agent calls an event as it would a static method. Agents who have subscribed to this event can respond in some way.

A delegate and event pair:

```
public delegate void someEvent();
public static event someEvent OnSomeEvent;
```

The agent can call the event when it wants to send a message. A second agent can subscribe and unsubscribe from the event:

```
sender.OnSomeEvent += doSomethingMethod;; //subscribe
sender.OnSomeEvent -= doSomethingMethod;; //unsubscribe
```