Table of Contents

State Machine

Usage

Create a class that is derived from StateMachine, which is a class in the StateMachineLanguage namespace.

```
using StateMachineLanguage;

public class ExampleStateMachine : StateMachine
{}
```

Declare state ID's (integers)

```
public class ExampleStateMachine : StateMachine
{
    const int STATE_Initialize = 0;
    const int STATE_WaitForBegin = 1;
    const int STATE_Animate = 2;
}
```

Make sure each integer is unique. State ID's must not change, so they should be declared constant.

Declare a state machine

Most often a state machine can be declared in the Unity Awake() function, but sometimes it can be declared in Unity's Start() function or some other initialize function. The following is an example of a class that fully declares a state machine:

```
using StateMachineLanguage;
public class CameraShakeActor : StateMachine
{
    const int STATE_Initialize = 0;
    const int STATE_WaitForBegin = 1;
    const int STATE_Animate = 2;

    private void Awake()
    {
        BeginDeclaration(STATE_Initialize);

        DeclareState(STATE_Initialize, delegate (State state)
        {
            OnEnter(delegate ()
            {
                ChangeState(STATE_WaitForBegin);
            });
        });

        DeclareState(STATE_WaitForBegin, delegate (State state)
        {
            OnMsg(MsgID.MSG_Begin, delegate ()
            {
                ChangeState(STATE_Animate);
            });
        });

        DeclareState(STATE_Animate, delegate (State state)
        {
            OnEnter(delegate ()
            {
                // Trigger the animation
            });

            OnMsg(MsgID.MSG_End, delegate ()
            {
                ChangeState(STATE_WaitForBegin);
            });
        });

        EndDeclaration();
    }
}
```

The state machine is declared between the function calls BeginDeclaration() and EndDeclaration(). Between the functions, any number of states can be declared. Each state is identified by an integer that is declared above, and the function BeginDeclaration() takes a state ID as an argument. The state ID that is passed to BeginDeclaration() is used to choose which state the state machine will start in. In the example above, the starting state is `STATE_Initialize`.

Declare states

A state is declared between BeginDeclaration() and EndDeclaration() using the function DeclareState(). The function DeclareState() takes the state ID as the first argument. The body of a state can contain blocks of code used for processing the state. Each block can be thought of as an event. The following is a list of each event that a state can contain:

State events

OnEnter()

This is the first block of code to execute when a state is entered. A state should contain no more than one of these.

OnExit()

This is the last block of code to execute when a state is exited. A state should contain no more than one of these.

OnUpdate()

This block of code will execute once per frame while the state is running. A state may contain more than one of these.

OnTimeInState()

The first argument passed to this is a value that represents how long to wait before executing the block of code once. The block of code will execute only once after the amount of time, if the state is not left before the amount of time passed since the state was entered. Otherwise, the block of code will not execute. A state may contain more than one of these with different time arguments.

OnPeriodicTimeInState()

The first argument passed to this is a value that represents the interval of time to wait before executing the block of code. The block of code will execute once after the interval of time, and it will repeat at each time interval while the state is running. If the state is exited before the interval of time, the block of code will not execute. A state may contain more than one of these with different time arguments.

OnMsg()

The first argument passed to this is a message ID. The block of code will execute when a message is sent to the state machine while the state is running. A state may contain more than one of these with different message ID's. See the message system section for details about message passing and message ID's.

Changing states

ChangeState()

To change states, call the function ChangeState() with the ID of the state that is to be changed to. The following is an example of changing state from one to another:

```
DeclareState(STATE_ExampleState, delegate (State state)
{
    OnMsg(MsgID.MSG_Begin, delegate ()
    {
        ChangeState(STATE_SomeState);
    });

    OnTimeInState(2.5f, delegate ()
    {
        ChangeState(STATE_SomeState);
    });
});
```

In the example above, `STATE_ExampleState` will transition to `STATE_SomeState` if the message `MSG_Begin` is received in the state, or if the amount of time in the state reaches 2.5 seconds.

PopState()

A state change can occur using the function PopState(). PopState() will change from whatever the current state is to whatever the previous state was. Before calling PopState(), make sure that there is always a state to go back to. For example, calling PopState() from the starting state is invalid, because there is no previous state. Also, the maximum number of PopState() calls that can occur consecutively is 5.

State change notes

Be aware that code following a function call that changes the state may or may not be executed one last time before changing states. For example, see the following block of code in a state:

```
OnUpdate(delegate ()
{
     //Some code omitted

     currAnimTime += Time.deltaTime;

     if (currAnimTime > totalAnimTime)
         ChangeState(STATE_SomeState);

     Debug.Log("Reached the end of OnUpdate().");

});
```

When the condition `currAnimTime > totalAnimTime` is true and `ChangeState(STATE_SomeState)` is called, the lines of code that follow may or may not be called after. In other words, the `Debug.Log()` function may or may not be called when `currAnimTime > totalAnimTime` is true. One way to ensure that the `Debug.Log()` function is not called after changing states is to make an else statement like so:

```
OnUpdate(delegate ()
{
     //Some code omitted

     currAnimTime += Time.deltaTime;

     if (currAnimTime > totalAnimTime)
         ChangeState(STATE_SomeState);
     else
         Debug.Log("Reached the end of OnUpdate().");
});
```

Scope of variables

Variables declared in a block of code in a state are local to the block of code. For example, see the following:

```
public class ExampleStateMachine : StateMachine
{
    //Some code omitted

    private float currAnimTime = 0.0f;
    private float totalAnimTime = 0.3f;

    private void Awake()
    {
        BeginDeclaration(STATE_Initialize);

        //Some code omitted

        DeclareState(STATE_Animate, delegate (State state)
        {
            OnUpdate(delegate ()
            {
                float t = currAnimTime / totalAnimTime;
            });
        });

        EndDeclaration();
    }
}
```

The variable t declared in the OnUpdate() block is local to the OnUpdate() block. To make a variable that is accessible to all states of a state machine, declare it in the state machine class. The variables currAnimTime and totalAnimTime above are declared in the state machine, and are accessible to all states of the state machine.

Message System

MsgID

To send a message, the message ID (integer) must be declared in the MsgID class in the MessageSystem.cs script. The following are some example MsgID declarations:

```
public static class MsgID
{
    public const int MSG_Reset = 0;
    public const int MSG_Begin = 1;
    public const int MSG_End = 2;
    public const int MSG_Pause = 3;

    //declare more message ID's here if needed
}
```

The following are a few ways to send a message to a state machine's state:

SendMsgToState()

```
void SendMsgToState(int msgID)
```

Use this function to send a message to the current state of a state machine when the message is being sent from within the state machine that is to receive the message, or if the StateMachine object is available such that SendMsgToState() can be called as a member function. The argument is the MsgID that is to be sent.

SendMsg()

```
static void SendMsg(GameObject target, int msgID)
```

Use this function to send a message to the current state of a state machine when the GameObject with the StateMachine component that is to receive the message is available. The first argument is the GameObject with the attached StateMachine component that is to receive the message. The second argument is the MsgID that is to be sent.

SendMsgToStateDelayed()

```
void SendMsgToStateDelayed(int msgID, float delay)
```

This is the same as the SendMsgToState() function described above except SendMsgToStateDelayed() takes an additional parameter that is an amount of time to wait before sending the message.

SendMsgDelayed()

```
void SendMsgDelayed(GameObject target, int msgID, float delay)
```

This is the same as the SendMsg() function described above except SendMsgDelayed() takes an additional parameter that is an amount of time to wait before sending the message.