StateMachines

This packages offers astract classes for **FSM** (***Finite State Machine***) and **HFSM** (***Hierarchical Finite State Machine***) patterns.

Do not waste time with all the state machine mechanics anymore, just implement the logic!

You can even save more time by using the scripts generator (**Window > FSM/HFSM Generator**)

Complete API Documentation

Usages

FSM

Here is a simple example of a state machine with three states A, B and C.

```
public class DemoStateMachine : AbstractFiniteStateMachine
{
    // Declaring the states Enum
    public enum DemoState
    {
        A,
        B,
        C
    }
    // Before Start
    private void Awake()
    {
        // Initializes the state machine with the default state enum, then all the states (ord
        Init(DemoState.A,
            AbstractState.Create<AState, DemoState>(DemoState.A, this),
            AbstractState.Create<BState, DemoState>(DemoState.B, this),
            AbstractState.Create<CState, DemoState>(DemoState.C, this)
        );
    }
    // Declaring the states
    public class AState : AbstractState
    {
        // Will be called on state entry
        public override void OnEnter()
        {
        }
        // Will be called on state update
        public override void OnUpdate()
        {
            // On any input
            if (Input.anyKeyDown)
            {
                // Will transition to the state B
                TransitionToState(DemoState.B);
            }
        }
        // Will be called on state exit
        public override void OnExit()
        {
        }
    }
    public class BState : AbstractState
    {
        public override void OnEnter()
        {
        }
        public override void OnUpdate()
        {
            if (Input.anyKeyDown)
            {
                TransitionToState(DemoState.C);
            }
        }
        public override void OnExit()
        {
        }
    }
    public class CState : AbstractState
    {
        public override void OnEnter()
        {
        }
        public override void OnUpdate()
        {
            if (Input.anyKeyDown)
            {
                TransitionToState(DemoState.A);
            }
        }
        public override void OnExit()
        {
        }
    }
}
```

you can also declare and use the OnFixedUpdate method inside your states

```
public override void OnFixedUpdate()
{
}
```

HFSM

Here is a simple example of a state machine with three states A, B and C, with the B state being also a nested state machine with its own states (SUB_A, SUB_B and SUB_C)

- MainStateMachine.cs

```
public class MainStateMachine : AbstractHierarchicalFiniteStateMachine
{
    // Declaring the state machine states Enum
    public enum MainState
    {
        A,
        B,
        C
    }
    public MainStateMachine()
    {
        // Initializes the state machine with the default state enum, then all the states (ord
        Init(MainState.A,
            Create<AState, MainState>(MainState.A, this),
            Create<BStateMachine, MainState>(MainState.B, this),
            Create<CState, MainState>(MainState.C, this)
        );
    }
    // Will be called when exiting from a sub state machine
    // If you do not override this method, the base one will trigger a transition to the defau
    // Be aware that if you do override this method and do not make a transition happen then y
    public override void OnExitFromSubStateMachine(AbstractHierarchicalFiniteStateMachine subS
    {
        TransitionToState(MainState.C);
    }
    // Declaring the simple states (that are not a sub state machine)
    public class AState : AbstractState
    {
        public override void OnEnter()
        {
        }
        public override void OnUpdate()
        {
            if (Input.anyKeyDown)
            {
                TransitionToState(MainState.B);
            }
        }
        public override void OnExit()
        {
        }
    }
    public class CState : AbstractState
    {
        public override void OnEnter()
        {
        }
        public override void OnUpdate()
        {
            if (Input.anyKeyDown)
            {
                TransitionToState(MainState.A);
            }
        }
        public override void OnExit()
        {
        }
    }
}
```

- BStateMachine.cs

```
public class BStateMachine : AbstractHierarchicalFiniteStateMachine
{
    public enum SubState
    {
        SUB_A,
        SUB_B,
        SUB_C
    }
    public BStateMachine()
    {
        Init(SubState.SUB_A,
            Create<SubAState, SubState>(SubState.SUB_A, this),
            Create<SubBState, SubState>(SubState.SUB_B, this),
            Create<SubCState, SubState>(SubState.SUB_C, this)
        );
    }
```

```
// Will be called on the sub state machine entry
public override void OnStateMachineEntry()
{
}
// Will be called on the sub state machine exit
public override void OnStateMachineExit()
{
}
public class SubAState : AbstractState
{
    public override void OnEnter()
    {
    }
    public override void OnUpdate()
    {
        if (Input.anyKeyDown)
        {
            TransitionToState(SubState.SUB_B);
        }
    }
    public override void OnExit()
    {
    }
}
public class SubBState : AbstractState
{
    public override void OnEnter()
    {
    }
    public override void OnUpdate()
    {
        if (Input.anyKeyDown)
        {
            TransitionToState(SubState.SUB_C);
        }
    }
    public override void OnExit()
    {
    }
}
public class SubCState : AbstractState
{
    public override void OnEnter()
    {
    }
    public override void OnUpdate()
    {
        if (Input.anyKeyDown)
```

```
            {
                // Exiting the sub state machine (will make the parent one trigger a transitio
                TransitionToState(EXIT);
            }
        }
        public override void OnExit()
        {
        }
    }
}
```

The nested state machines do not inherit from MonoBehaviour, you cannot drop the root state machine as a component on an GameObject, you have to implement a "wrapper" component for it.

Here is an example of wrapper component for the above example.

- MainStateMachineComponent.cs

```
public class MainStateMachineComponent : MonoBehaviour
    {
        private MainStateMachine _stateMachine;
        private void Awake()
        {
            // Instantiate the root state machine through the CreateRootStateMachine method an
            _stateMachine = AbstractHierarchicalFiniteStateMachine.CreateRootStateMachine<Main
        }
        private void Start()
        {
            // Call OnEnter on your root state machine on the Start of the wrapper component
            _stateMachine.OnEnter();
        }
        private void Update()
        {
            // Call OnUpdate on your root state machine on the Update of the wrapper component
            _stateMachine.OnUpdate();
        }
        // Do the same for FixedUpdate if you need to
        private void FixedUpdate()
        {
            // Call OnFixedUpdate on your root state machine on the FixedUpdate of the wrapper
            _stateMachine.OnFixedUpdate();
        }
    }
```

Generators

You can use generators windows for both **FSM** and **HFSM** to generate skeleton, ready-to-implement scripts, along with some options.

Open these windows via the menu ***Window*** > ***FSM/HFSM Generator***

Screen1 Screen2 Screen3