State Machines

What are they, When to use them, and How to make them

Nathan Clark
Local Unity Developer, he/him

What is a State Machine?

Finite State Machine (FSM)

- Model of behavior

A state machine is made up of:

- States
  - Condition of a system
- Transitions
  - How the state of a system changes

What is State and what does it do?

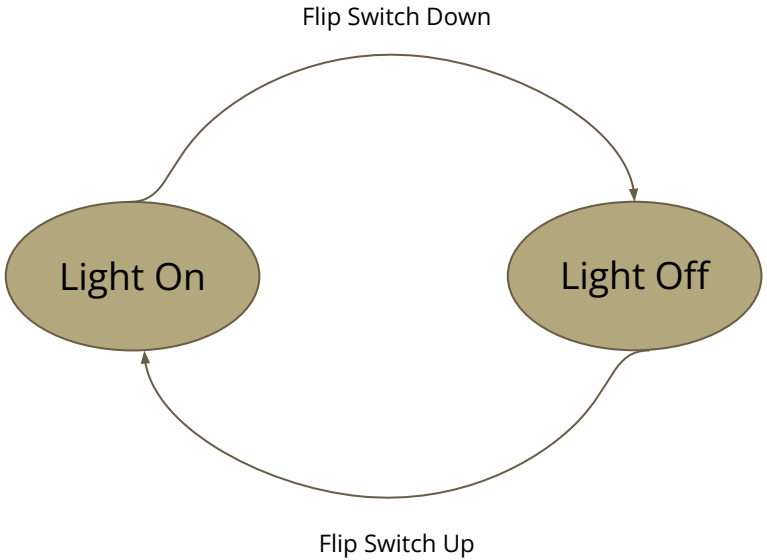
State is the description of the condition of a system

- jumping, running, climbing, swimming, etc.
- Angry, Scared, Asleep, Happy, Sad, etc.
- A system is only in one state at a time

What does state do?

- Defines what behavior is going to be running
  - Scared => runs away
  - Angry => attack player

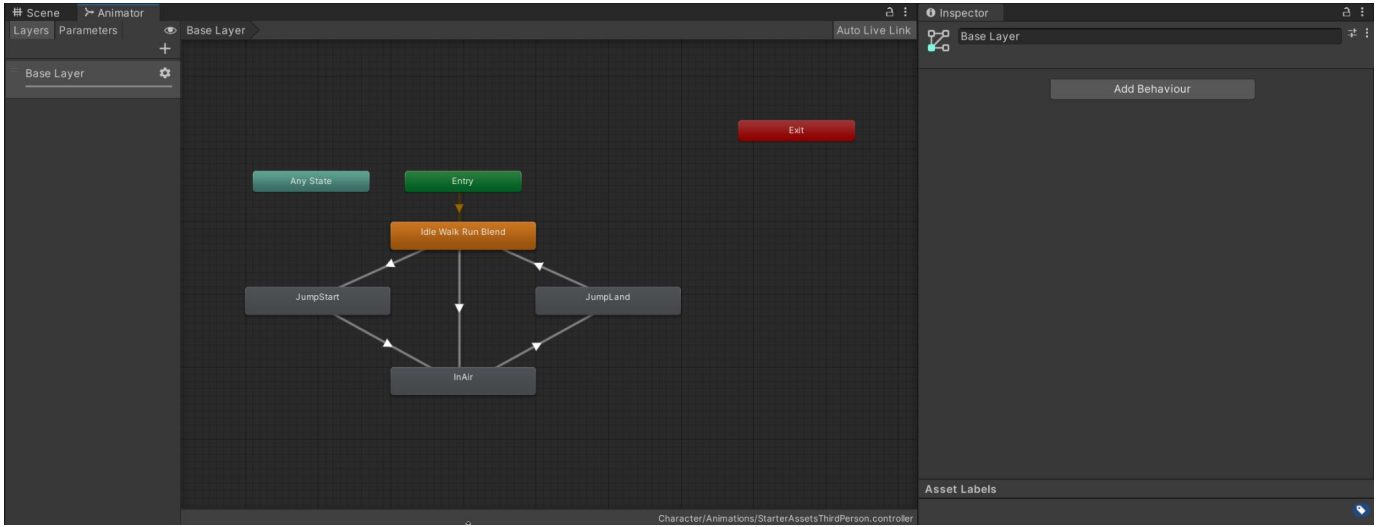
Example State Machine Diagram



Does this look familiar?

Unity Animator

The Animator is an example of a state machine



Why do we use State Machines?

Compartmentalize behavior

Example pseudocode for Character Controller

This is simple and it will work, but what happens when we add more states.

Why do we use State Machines?

Compartmentalize behavior

Example pseudocode for Character Controller

```
1  if(JumpInput && Grounded && !IsCrawling)
2      PerformJump()
3
4  if(IsCrawling)
5      speed = .5f;
6  else
7      speed = 1f;
8
9  PerformHorizontalMovement(speed);
```

This is simple and it will work, but what happens when we add more states.

Why do we use State Machines?

Compartmentalize behavior

Example pseudocode for Character Controller

```
1   if(JumpInput && Grounded && !IsCrawling && !Climbing)
2       PerformJump()
3
4   if(IsCrawling)
5       speed = .5f;
6   else
7       speed = 1f;
8
9   if(IsClimbing)
10      PerformClimbingMovement();
11  else
12      PerformHorizontalMovement(speed);
```

This is simple and it will work, but what happens when we add more states.

Things can get complex very quickly.

If statements scattered throughout

Why do we use State Machines? (cont.)

Jump State

```
PerformHorizontalMovement(1f);
```

Crawl State

```
PerformHorizontalMovement(.5f);
```

Grounded State

```
if(JumpInput)
    PerformJumpInput();

PerformHorizontalMovement(1f);
```

Climb State

```
PerformClimbingMovement();
```

How do we implement a State Machine?

Making a state machine is like cooking an egg

- Lots of ways to do it
- The easy ways don’t always taste great, and the hard ways aren’t always worth it

Minimum Requirements

- States
- Transitions

Enums and Switch Statements (or if else if)

The easiest way to make a state machine

Flawed

- Long, bloated scripts
- No reusability

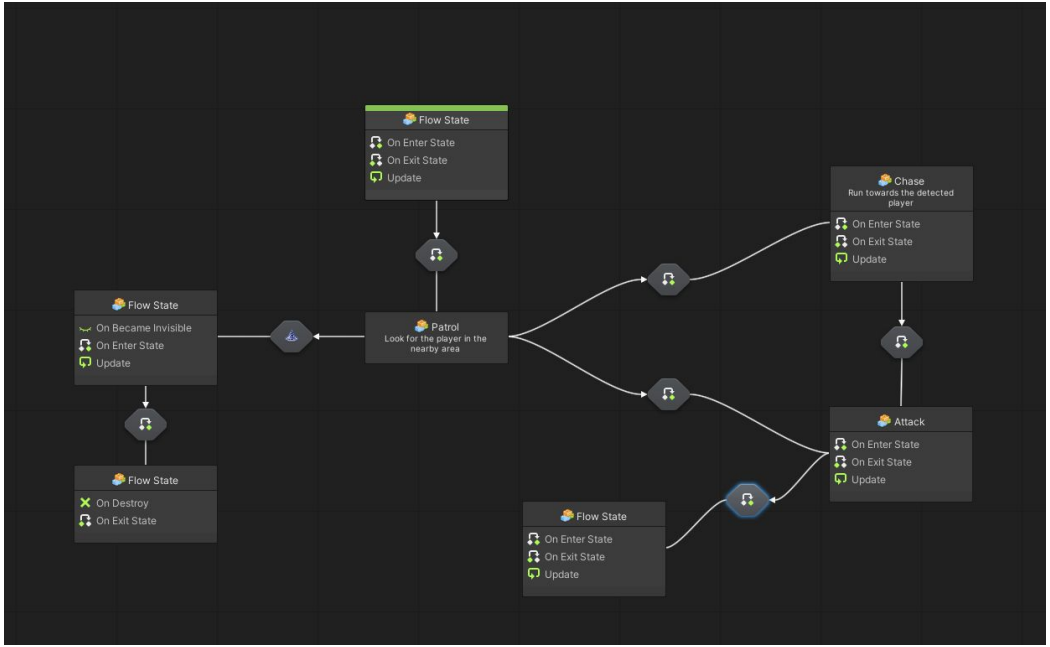
Use it when you need something simple

And quick

```
//State Update
switch (CurrentState)
{
    case SlimeState.Move:
        break;
    case SlimeState.Eat:
        float eatenMass = DigestionRate * Time.deltaTime;
        foreach (Fruit fruit in FruitInsideSlime)
        {
            if (fruit != null)
            {
                SizeController.Size += fruit.Eat(eatenMass);
            }
        }
        break;
    case SlimeState.Reproduce:
        transform.GetChild(0).gameObject.SetActive(false);
        Vector3 leftSpawnPosition = transform.position + (transform.forward * .5f) + (transform.right * .5f);
        leftSpawnPosition.y = .5f;

        Vector3 rightSpawnPosition = transform.position - (transform.forward * .5f) - (transform.right * .5f);
        rightSpawnPosition.y = .5f;
        SlimeSpawner.SpawnSlime(leftSpawnPosition, transform.rotation);
        SlimeSpawner.SpawnSlime(rightSpawnPosition, transform.rotation);
        break;
    case SlimeState.Death:
        Destroy(this.gameObject);
        break;
}
```

Visual Scripting Package



Using Components as State

Only slightly more difficult than switch statement

StateMachine component that enables and disables State components based on the current state

Reusable?

Potential problems

My opinion

Not worth it

Practical Implementation Time

My preferred way to make State Machines

Interfaces and ScriptableObjects

Ways to Improve

Reusability Issues

- Transitions inside of State
- States only accept SlimeStateMachine

What Functionality Belongs Inside StateMachine

Things we could’ve added

Any State Transitions

On Enter, On Exit

Really useful, Easy to implement, but wasn’t necessary for the Slime

Things we could've added (cont.)

Sub State Machine

- Basically a more complex state with it's own internal State Machine to determine it's behavior

Composite State

- State that combines multiple behaviors
- For example, a Composite State that has both a MoveState and EatState
  - Move and Eat at the same time
- Potentially Dangerous - use with caution

Final Thoughts

Time Sink

Add features as needed

State Machines are a good tool

- Learn to know when to use a hammer and not a drill

Questions?

https://github.com/ndclark94/StateMachines