Model Based Testing
a 3D game engine
Simple game object physics
Model Actions

- **SetMass (float)**

  Sets mass of rigid body. Higher mass requires higher force to move it.

- **AddForce (vector, mode)**

  Adds specified force (vector) to object, in order to make it move.
Force Modes

- **Force**
  Add a continuous force to the rigidbody, using its mass.

- **Acceleration**
  Add a continuous acceleration to the rigidbody, ignoring its mass.

- **Impulse**
  Add an instant force impulse to the rigidbody, using its mass.

- **VelocityChange**
  Add an instant velocity change to the rigidbody, ignoring its mass.
A Basic Physics Model

Mass=1, Velocity=(-20.0, 0.0, 0.0)

AddForce((10.0, 0.0, 0.0), VelocityChange)  AddForce((-10.0, 0.0, 0.0), VelocityChange)

Mass=1, Velocity=(-10.0, 0.0, 0.0)

AddForce((10.0, 0.0, 0.0), VelocityChange)  AddForce((-10.0, 0.0, 0.0), VelocityChange)

Mass=1, Velocity=(0.0, 0.0, 0.0)

AddForce((10.0, 0.0, 0.0), VelocityChange)  AddForce((-10.0, 0.0, 0.0), VelocityChange)

Mass=1, Velocity=(10.0, 0.0, 0.0)

AddForce((10.0, 0.0, 0.0), VelocityChange)  AddForce((-10.0, 0.0, 0.0), VelocityChange)

Mass=1, Velocity=(20.0, 0.0, 0.0)
Model Implementation using Spec Explorer

```csharp
[Rule]
public static void AddForce([Domain("ForcePower")]) Vector3 force, [Domain("ForceModeValue")]) ForceMode forceMode)
{
    const float fixedDeltaTime = 0.02f; // = 50 FPS

    switch (forceMode)
    {
        case ForceMode.Acceleration:
            ModelState.Velocity += force * fixedDeltaTime;
            break;
        case ForceMode.Force:
            ModelState.Velocity += force * fixedDeltaTime / ModelState.Mass;
            break;
        case ForceMode.Impulse:
            ModelState.Velocity += force / ModelState.Mass;
            break;
        case ForceMode.VelocityChange:
            ModelState.Velocity += force;
            break;
    }
}

[Rule]
public static void SetMass([Domain("Mass")]) float mass)
{
    Condition.IsFalse(ModelState.Mass.Equals(mass));

    ModelState.Mass = mass;
}
```
Limiting the Model Outcome State Space
Unity Runtime Test Framework

Test Case

Runtime Test Framework

Runtime Players

Windows
Mac
Linux
iPhone
Android
BlackBerry

plus more ...
Connecting the Model and Framework

Model / Spec Explorer

C# Unit Test Code

Runtime Test Code

Runtime Test Framework

Unity Runtime
Connecting the Model and Framework - solution

- Spec Explorer generates C# unit tests
- Runtime Test Framework requires one class per test case
- Dynamic code generation from model generated test cases