# Model Based Testing of a 3D game engine

Rasmus Møller Selsmark
Team Lead, Infrastructure
rasmuss@unity3d.com

## Mission Statement

How can Model Based Testing be applied to a game engine?

Experiences and findings from using Microsoft Spec Explorer for model based testing of

- Game editor: Version Control Integration
- Runtime: Simple in-game physics

## About Unity

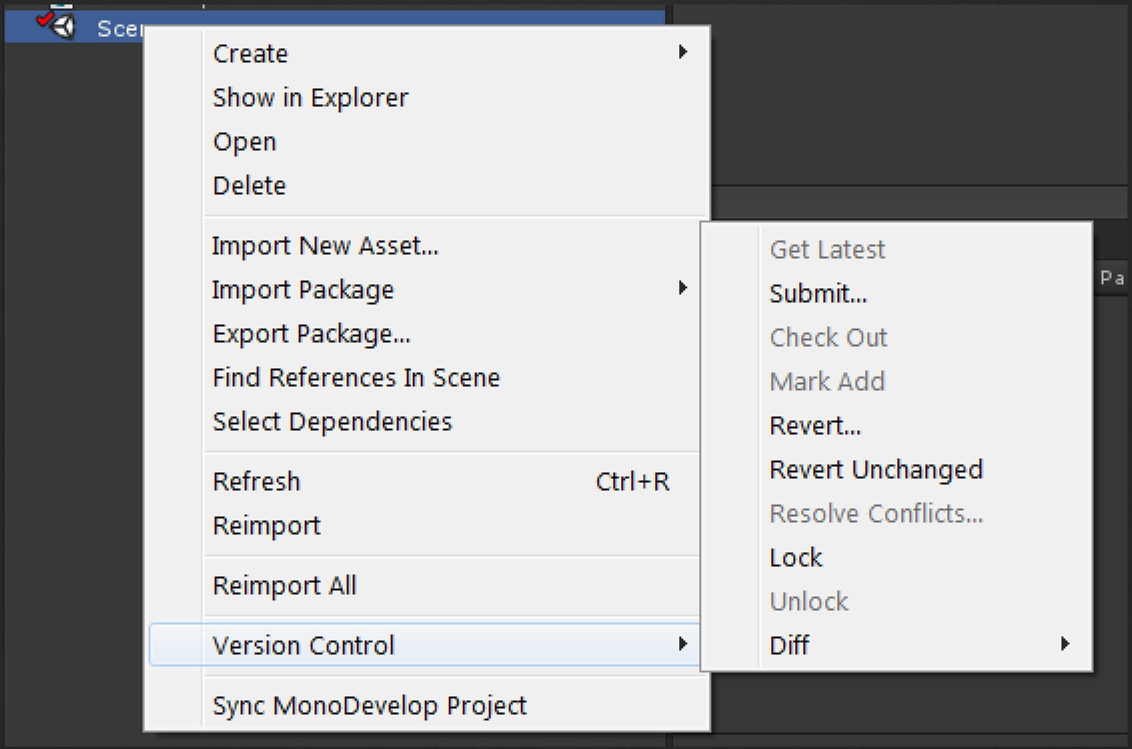
- Environment for developing games for PC, Mac, Web, mobile and consoles
- 2M registered/400K active game developers
- 225M installed web players
- Uses Mono cross-platform scripting engine

Some test and automation numbers:

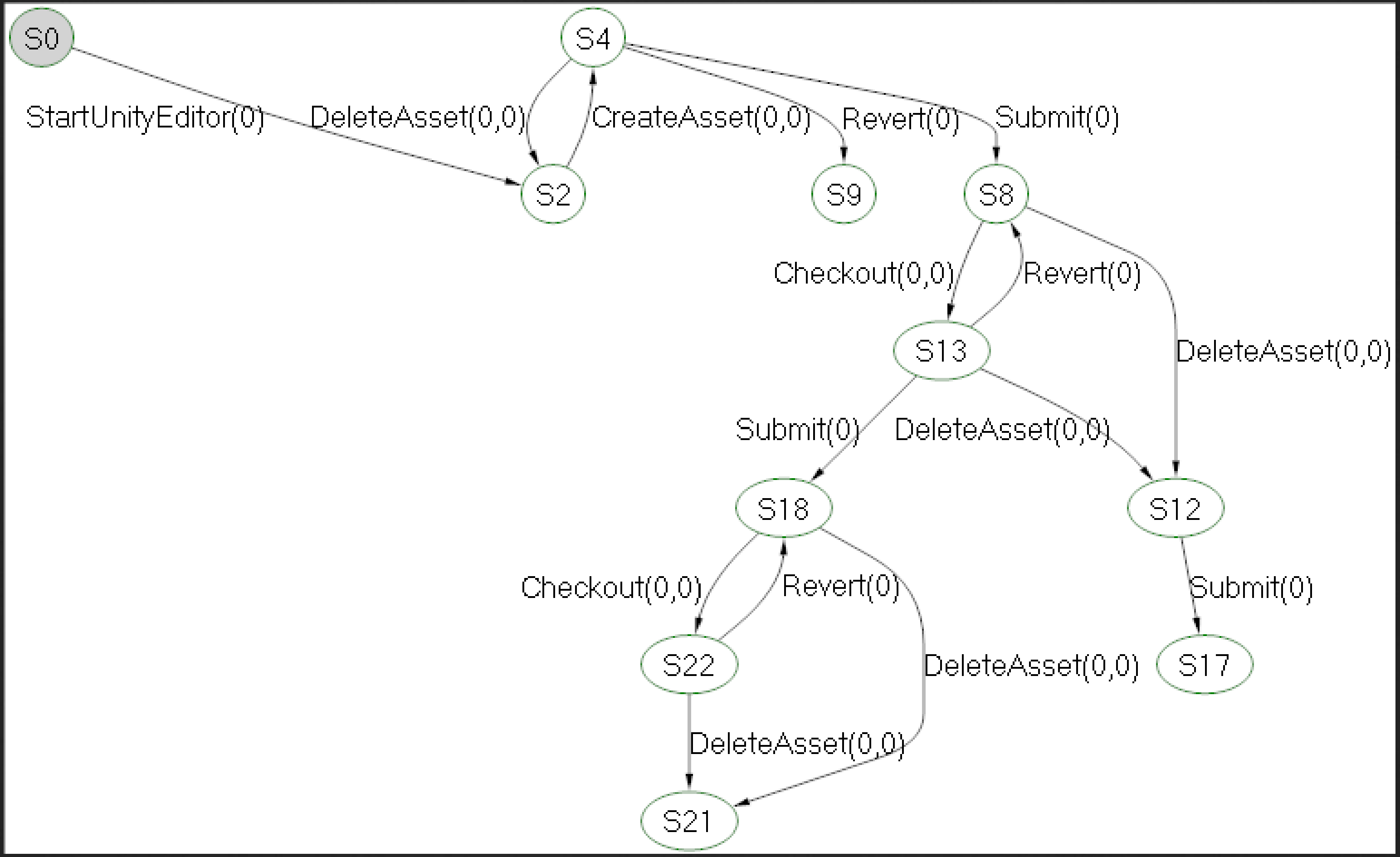
- 14 devs working fulltime on framework/tests
- 1050 runtime tests executed on 21 different platforms (“game” tests)
- 1150 integration tests run on Win+Mac (editor)

## Version Control Integration

- New feature
- Critical area (bugs can cause data loss)



# Base Version Control Integration Model



## Version Control Integration - Findings

- Number of tests executed: 706
- Bugs were found during modeling phase, but not during execution
- Unit testing the model has been valuable for validating correct behavior of model and actions and when adding new actions
- Model based testing applies well to domain
- Spec Explorer fits into implementation of this area in Unity

## Modeling in-game physics

- Physics is an important area in a game
- Pilot project for applying MBT to runtime tests

**Force = mass * acceleration**
(Newtons second law)

**Unity GameObject method:**

Rigidbody.AddForce(**force**: Vector3,
  **mode**: ForceMode);

Adds a force to the rigidbody. As a result the rigidbody will start moving

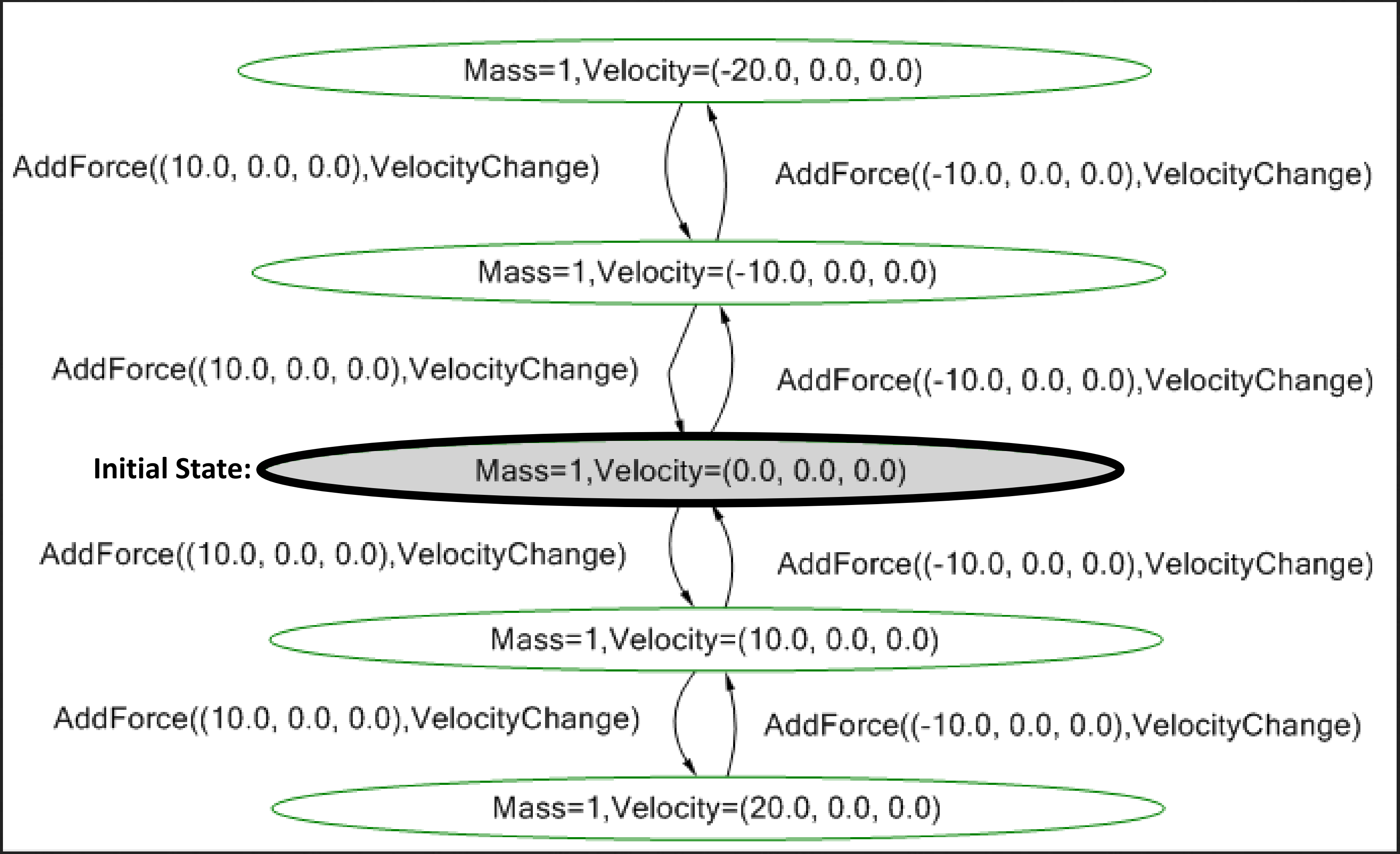
## Demo: AddForce modes

Force
Acceleration
Impulse
VelocityChange
DBFA 12
DBFA 12

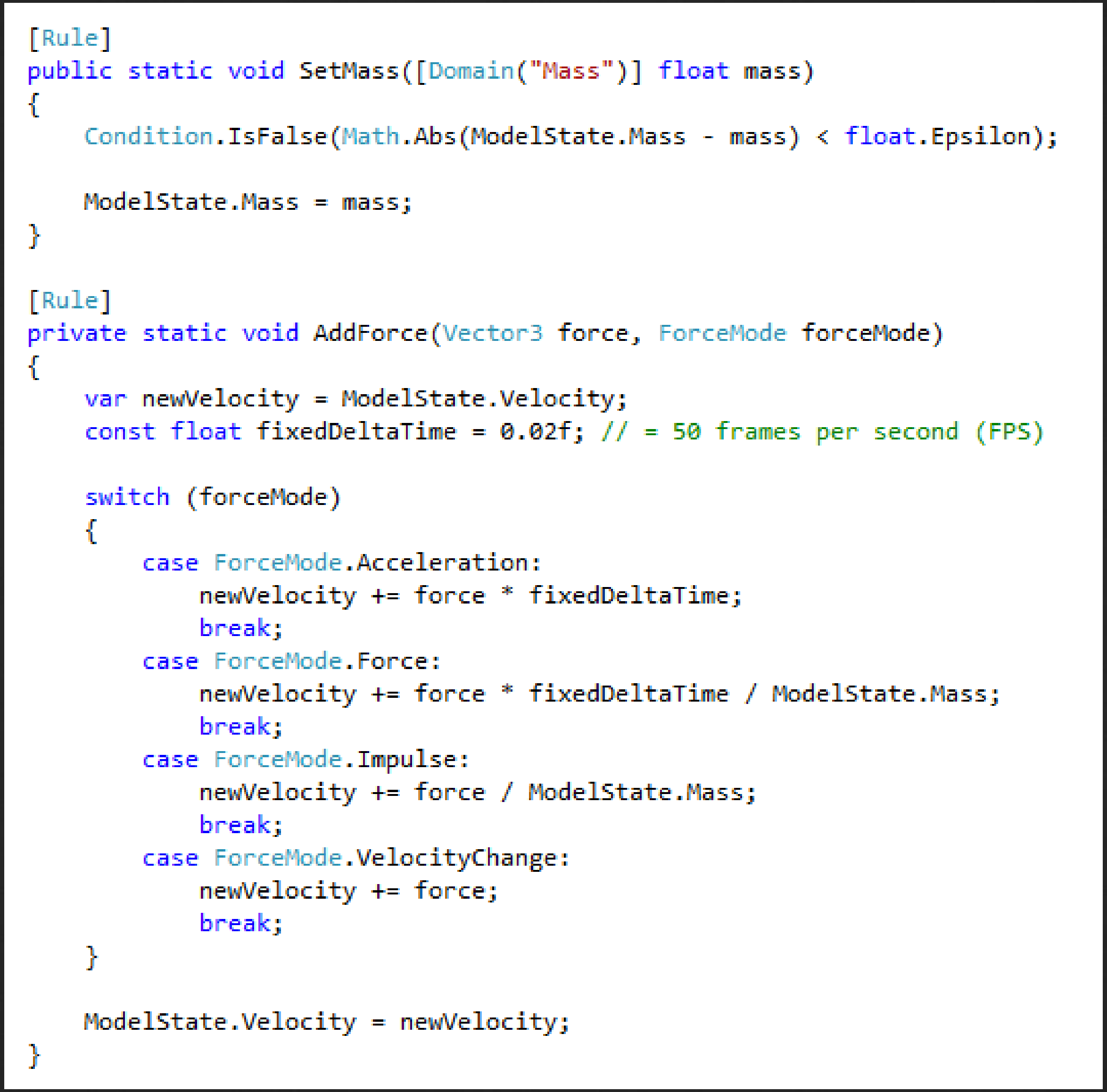
## Description of Force Modes

- **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 Very Simple Physics Model (no gravity)

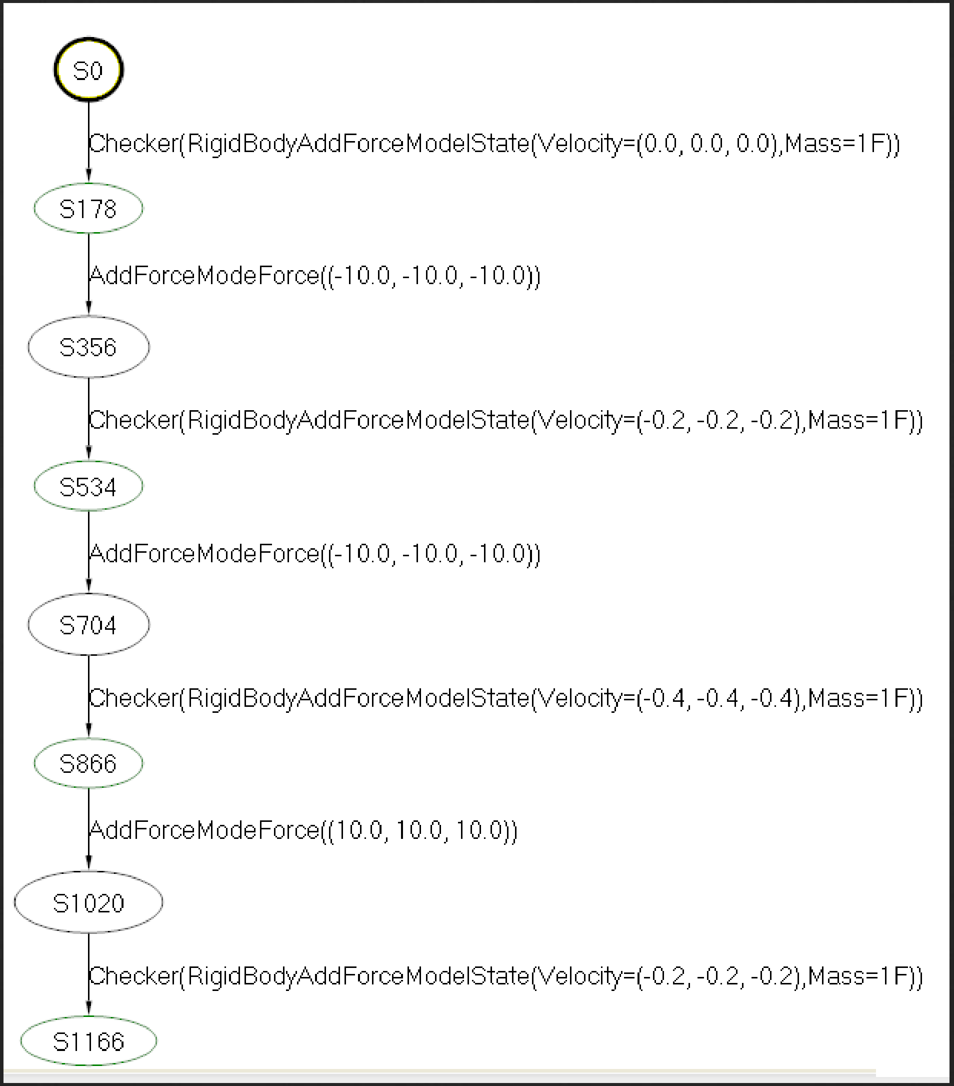

Mass=1,Velocity=(-20.0, 0.0, 0.0)
AddForce((10.0, 0.0, 0.0),VelocityChange)
AddForce((-10.0, 0.0, 0.0),VelocityChange)
Mass=1,Velocity=(-10.0, 0.0, 0.0)
AddForce((10.0, 0.0, 0.0),VelocityChange)
AddForce((-10.0, 0.0, 0.0),VelocityChange)
Initial State:
Mass=1,Velocity=(0.0, 0.0, 0.0)
AddForce((10.0, 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



```
[Rule]
public static void SetMass([Domain("Mass")] float mass)
{
    Condition.IsFalse(Math.Abs(ModelState.Mass - mass) < float.Epsilon);

    ModelState.Mass = mass;
}

[Rule]
private static void AddForce(Vector3 force, ForceMode forceMode)
{
    var newVelocity = ModelState.Velocity;
    const float fixedDeltaTime = 0.02f; // = 50 frames per second (FPS)

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

    ModelState.Velocity = newVelocity;
}
```

# Demo: Spec Explorer generated Test Case


S0
Checker(RigidBodyAddForceModelState(Velocity=(0.0, 0.0, 0.0),Mass=1F))
S178
AddForceModeForce((-10.0, -10.0, -10.0))
S356
Checker(RigidBodyAddForceModelState(Velocity=(-0.2, -0.2, -0.2),Mass=1F))
S534
AddForceModeForce((-10.0, -10.0, -10.0))
S704
Checker(RigidBodyAddForceModelState(Velocity=(-0.4, -0.4, -0.4),Mass=1F))
S866
AddForceModeForce((10.0, 10.0, 10.0))
S1020
Checker(RigidBodyAddForceModelState(Velocity=(-0.2, -0.2, -0.2),Mass=1F))
S1166

## Game Physics - Findings

- Number of tests from model: 89
- No bugs found (already relatively high test coverage for this area)
- Model based testing does apply to domain of testing generic game engine behavior
- The tool (Spec Explorer) isn’t optimal in relation to “disconnected tests” requirement of our Runtime Testing Framework (for executing tests on e.g. mobile devices and consoles)

## Summary

- The **principles** of model based testing applies well to editor/back-end features, as well as some areas of run-time tests
- Need to either solve problem of generating “disconnected tests” using Spec Explorer or look at other MBT frameworks for runtime tests
- We will continue developing model based tests, with the goal of increasing coverage of oucritical areas of application
- Most likely focus on editor/back-end features like Undo system, license etc