Introduction to UrhoSharp

RYAN DAVIS

Queensland C# Mobile Developers Meetup

2017 10 24

whoami

- Ryan Davis
- Hobby Developer, .NET and mobile
- ~~Xamarin~~ Microsoft MVP

 ryandavis.io  rdavis_au  rdavisau

to cover

- Overview of UrhoSharp
- Walk through the framework
- Samples and Demos

-= introduction to urhosharp =-

what is it?

a cross-platform 3D (and 2D) game engine for .NET

*"UrhoSharp is a **lightweight Game Engine** suitable for using with **C#** and **F#** to create **games** and **3D** applications […] powered by **Urho3D** (github readme)*

- Urho3D – a 3D game engine:
  - 3D rendering, scenes, physics, audio, declarative logic
  - built upon common tooling/frameworks – Bullet, box2d, Open Assets Library etc.

- UrhoSharp is a .NET binding to Urho3D
  - write your code in C# or F# (yay); async friendly api
  - bootstrap per platform + shared codebase (like forms)
  - access to the .NET ecosystem and features
  - bindings is open source and hosted on github

wpf
windows forms
uwp
android
h o l o l e n s
ios
osx
xamarin forms

a cross-platform 3D (and 2D) game engine for .NET

*"UrhoSharp is a **lightweight Game Engine** suitable for using with **C#** and **F#** to create **games** and **3D** applications […] powered by **Urho3D** (github readme)*

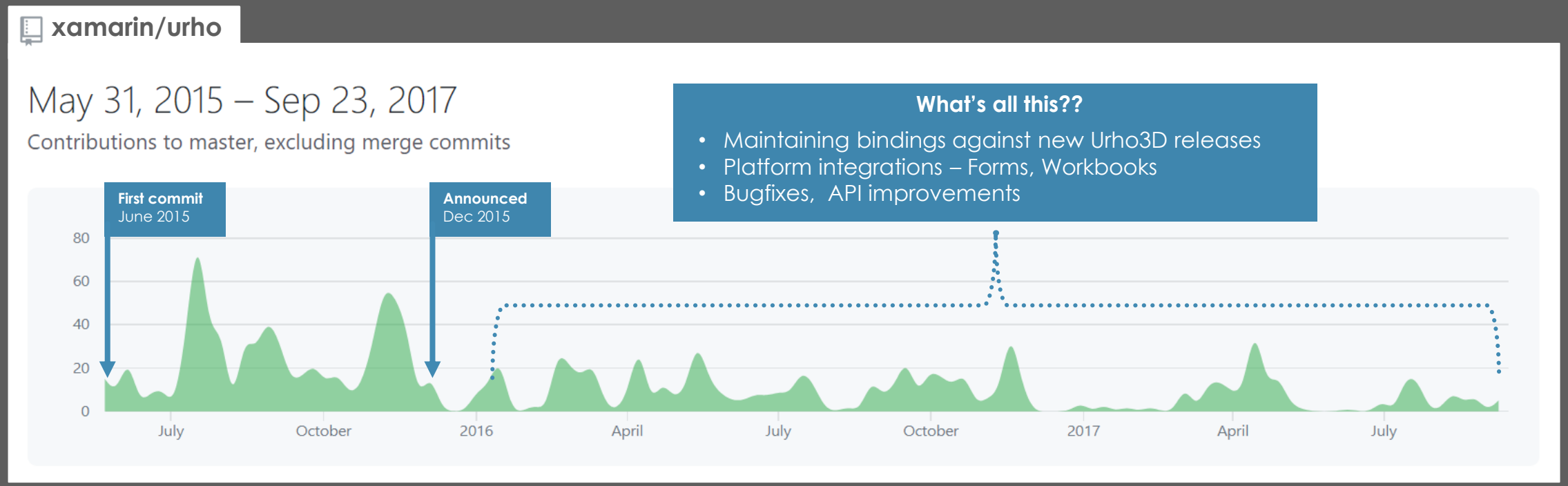
- the benefit of being "just" a wrapper over the Urho3D engine
  - based on a game framework that has been in development for ~10 years
  - you aren't limited to Xamarin/UrhoSharp documentation – examples for Urho3D will apply!

- "value add"
  - through saner c# API,
  - integration with Xamarin products

pictured: actual Urho3D logo

timelines

- UrhoSharp has been around for a while
  - Bindings developed during second half of 2015 and released late 2015
  - Still active in 2017.

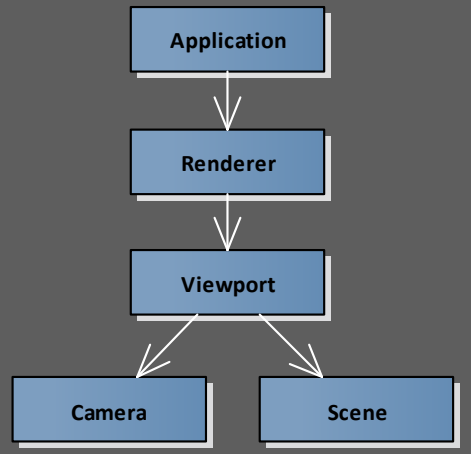


-= introduction to urhosharp =-

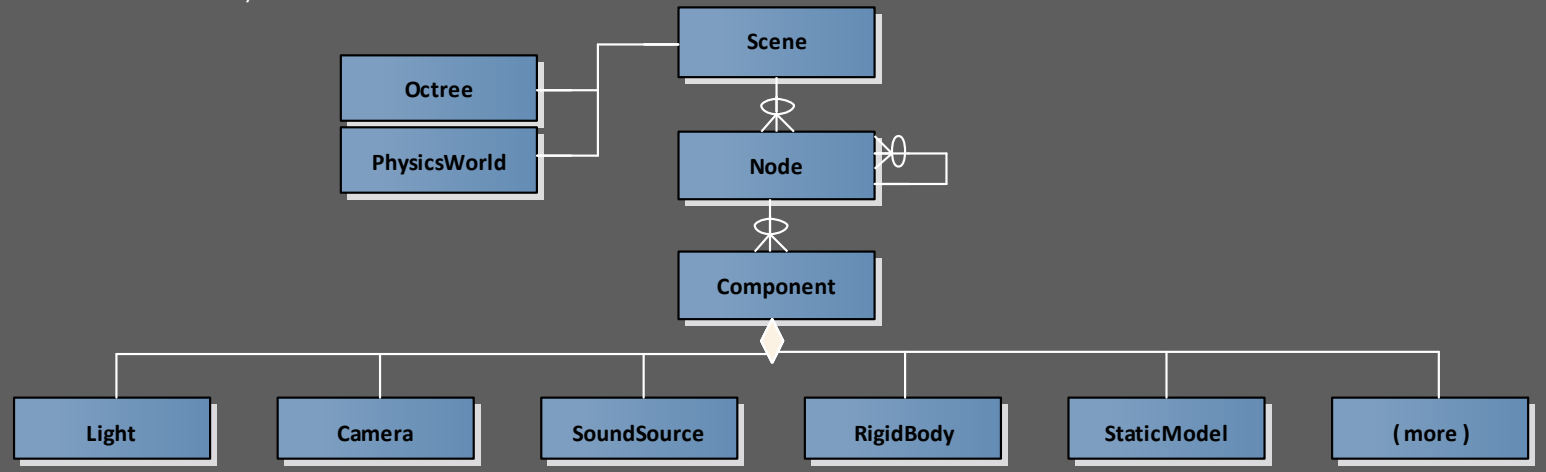
the framework

framework class structure (abridged)

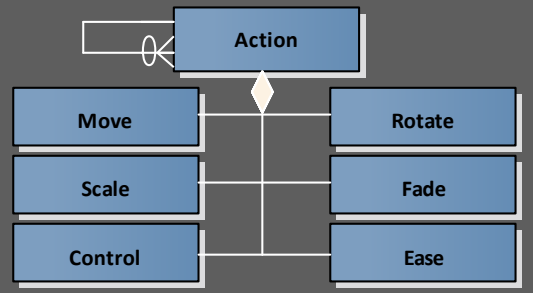
Application and Rendering



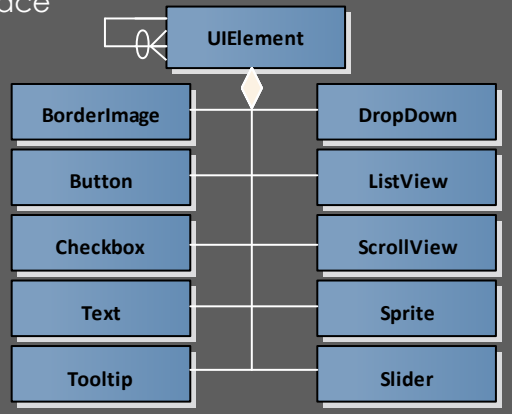
Scene/Node Hierarchy



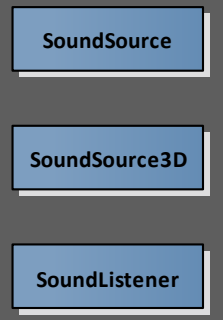
Animation and transforms



User Interface



Audio



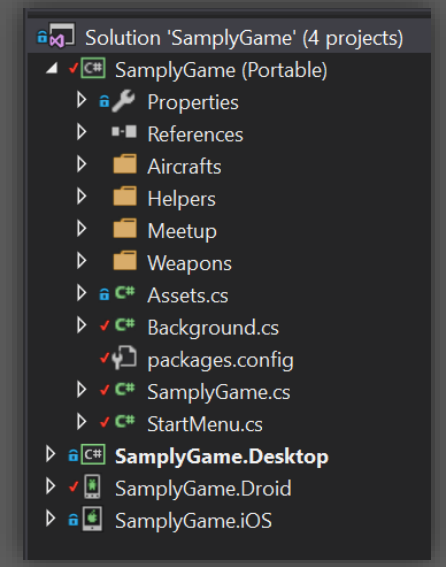
Physics

an urhosharp application

- Your game derives from the `Application` class, which includes a `Renderer` and typical lifecycle callbacks
- Your `Application` and game logic live in shared code
- Two approaches to running the game:
  - You can take an 'embedded' approach, running the UrhoSharp component/s in a section or view of your app - UrhoSharp provides 'Surface' classes that derive from native views, e.g. `UIView` on iOS and `Fragment` on Android.

OR

  - Your native projects can be limited to the 'Bootstrap' code necessary to transfer control to the UrhoSharp engine (similar to a Xamarin forms app).



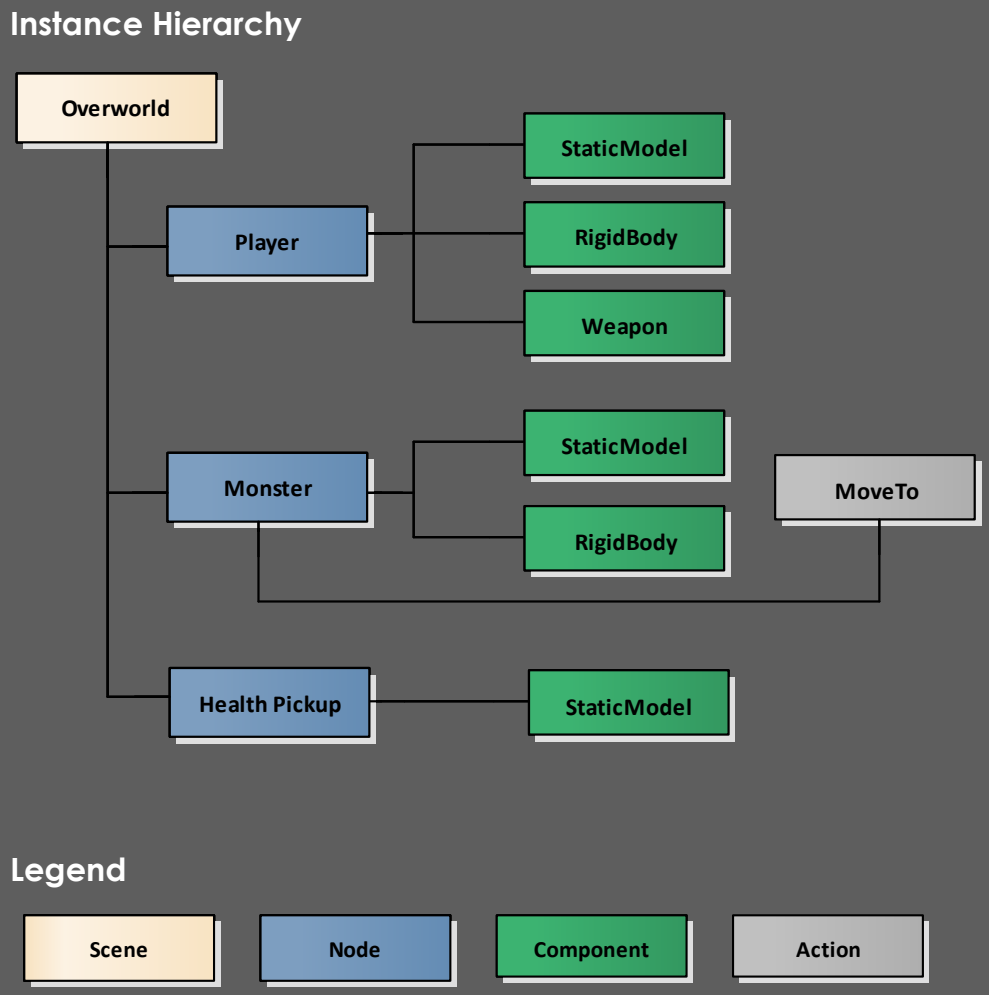
Typical project structure

```
class Program
{
    static void Main(string[] args)
    {
        DesktopUrhoInitializer.AssetsDirectory = @"../../Assets";
        new SamplyGame().Run();
    }
}
```

Native bootstrap
(that's all!)

scene hierarchy

- A `Scene` is the root of a game 'world' and is rendered into a `Viewport` by a `Camera`
- A `Scene` contains a hierarchy of `Node`s
  - A `Node` has an identifier and a translation in 3D-space
  - A `Node` may contain child `Node`s
- A `Node` can contain `Component`s which add behaviour
  - `StaticModel` – allows the the node to be rendered
  - `RigidBody` – allows the node to participate in physics
  - `SoundSource` – allows the node to emit audio
  - `Light` – allows the node to emit light
- `Action`s can also be attached to a `Node`, allowing declarative specification of behaviour.

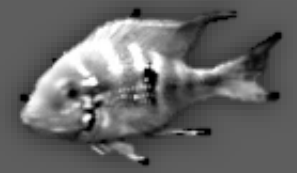


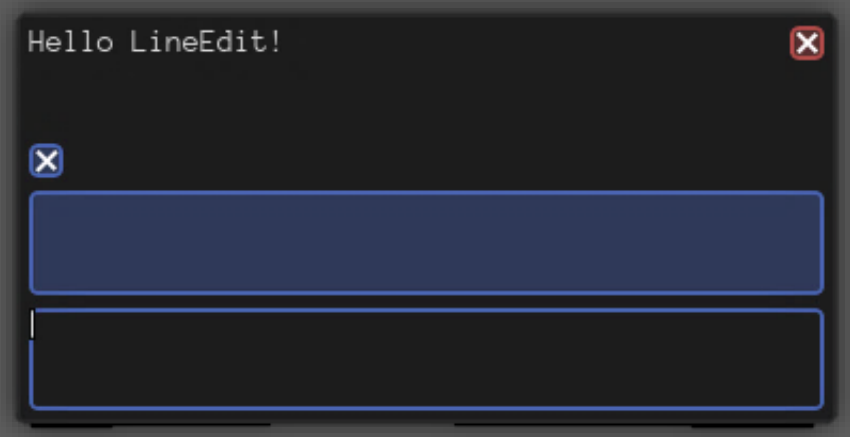
game logic – declarative or runloop

- Run an `Action` on a Node to declaratively specify behaviour and timing
  - Many actions included in the framework, or you can define your own
    - Translation: `MoveTo/By, BezierTo/By, RotateTo/By, ScaleTo/By`
    - Appearance: `FadeIn/Out/To, TintTo/By`
    - Control: `Reverse, Delay, Parallel, CallFunc,`
    - Interpolation: `InOut, Bounce, Sine, Elastic, Exponential`
  - Actions are designed to compose to allow specification of complex behaviours
    - `RepeatForever(EaseBounce(Sequence(FadeIn, FadeOut)))`
      [fade the node in then out indefinitely, with a bounce easing)]
- Perform work on a per-frame basis within your `Scene` and/or `Component` subclasses
  - In a `Scene`, register handlers for `SceneUpdate`, `SceneSubsystemUpdate` and/or `ScenePostUpdate`
  - In a `Component` subclass, set `ReceiveSceneUpdates` and override `OnUpdate`
- Using `async` adds another dimension to logic, simplifying otherwise complex scenarios.

user interface

- Subclasses of `UIElement` provide a basic UI capability to UrhoSharp games
- UI elements are rendered directly to the screen, and are not impacted by the scene hierarchy
- Many familiar controls are supported
  - `Text` (i.e. label)
  - `TextEntry`
  - `ListView`
  - `ScrollView`
  - `Slider`
  - `DropDownList`
  - `Button`
  - `CheckBox`
  - `Window`
  - `MessageBox`
- Some support for layout and alignment
- Add controls to `{AppInstance}.UI.Root`.





one two many dimensions?

- UrhoSharp is a 3D engine but it can still be used to create 2D games
- Call `SetOrthographic(true)` on the rendered `Camera` to give your 3D world a 2D representation
- Use the 2D-friendly classes and components for an optimised experience
  - UrhoSharp includes 2D sprite, spritesheet, texture, tile and particle effect classes
- box2d (used by many other game engines), is also used for physics in 2D UrhoSharp games.

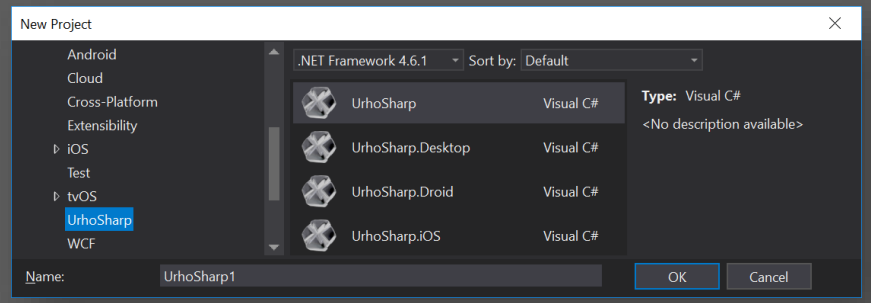
hololens

nuff said

(check out the UrhoSharp.SharpReality package on NuGet)

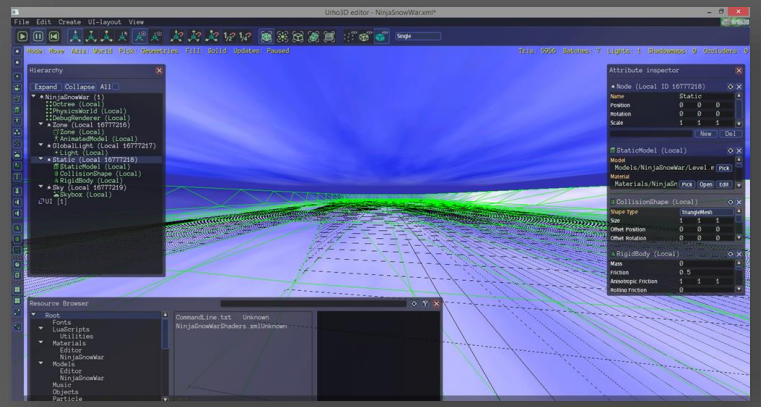
tooling support

- Project templates for VS [link]
  Cross-platform quickstart for iOS, Android and Desktop

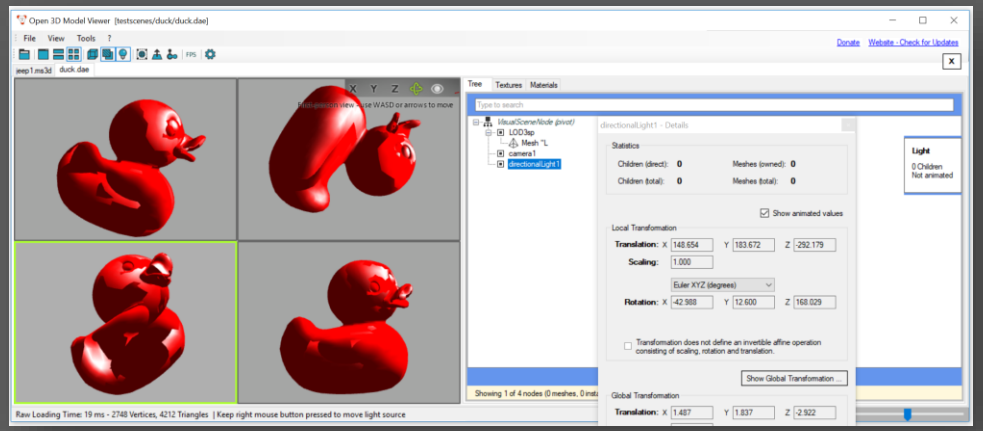


- Particle Designer [link] ($)(osx only)
  Design and preview particle effects using a GUI

- UrhoEditor [link]
  Design scene hierarchies using a GUI



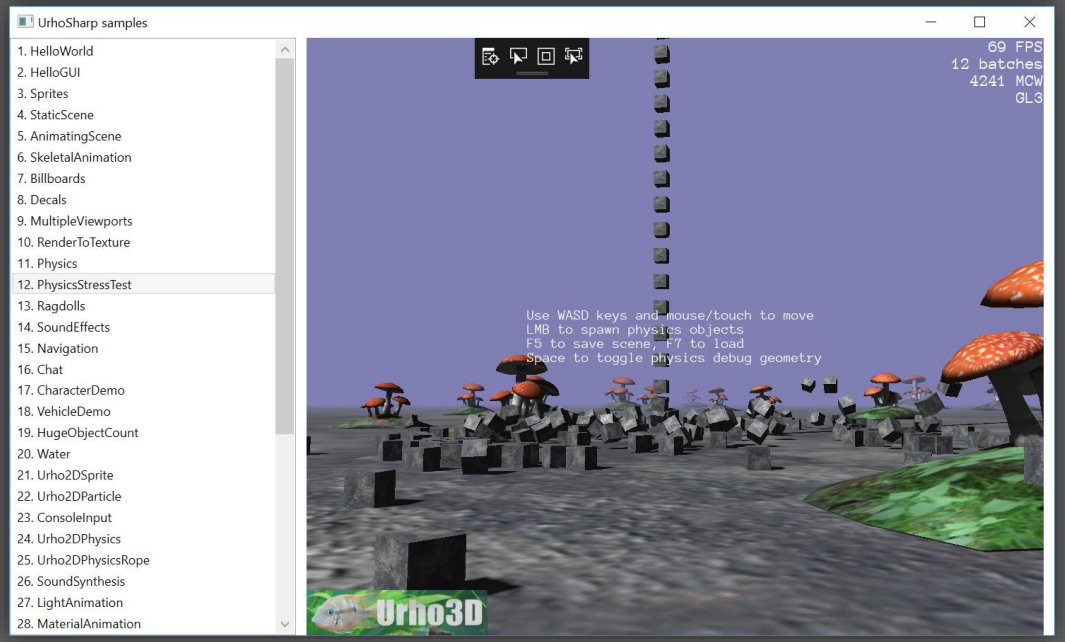
- Open3Mod [link]
  3D viewer and editor for model formats supported by Urho3D



Demo time

demos

Everything available at https://github.com/xamarin/urho-samples

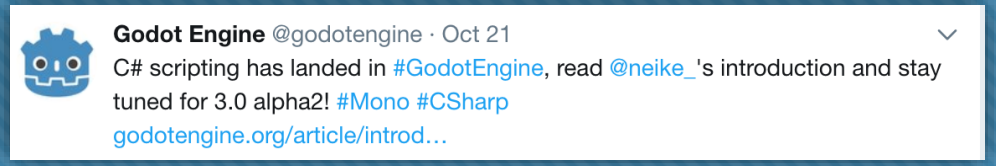


FeaturesSample

SamplyGame
(+ a few modifications)

closing

on choices



- In 2017, if you want to build a game and use C# you are spoilt for choice:

UrhoSharp

Unity3D

Unreal Engine

Xenko Engine

Atomic Game Engine
(Urho3D fork)

Wave Engine

Duality

MonoGame

- Each of these generally involves trade-offs across one of several dimensions – licensing, tooling, supported platforms, C# version, support, community, etc..

where next?

things I didn't cover

- **Networking**
  UrhoSharp has built in cross-platform networking, including low level transport and high level scene-replication functionality

- **Audio**
  Powered by libvorbis, UrhoSharp includes cross-platform audio support – 2D and "3D"

- **Physics**
  The Bullet and box2d physics libraries are used by UrhoSharp for 3D and 2D worlds respectively. The physics concepts and interaction methods will be familiar if you have used physics libraries in other engines

- **Serialisation and Prefabs**
  UrhoSharp has serialisation baked into the framework, allowing you to serialise or deserialise entire scene hierarchies for 'free' (note that any custom component subclasses will need to 'tell' the framework about the additional properties that should be serialised and handle the deserialisation of these only).

  Serialisation also allows you to implement the concept of 'prefabs' – predefined node hierarchies of objects you may want to spawn multiple times in your game.

useful resources

- **UrhoSharp Github**
  https://github.com/xamarin/urho
- **Xamarin UrhoSharp Documentation**
  https://developer.xamarin.com/guides/cross-platform/urho/
- **Urho3D page**
  https://urho3d.github.io/
- **UrhoSharp Samples**
  https://github.com/xamarin/urho-samples
- **UrhoSharp Workbooks**
  https://github.com/xamarin/Workbooks/tree/master/graphics/urhosharp

Install it using nuget!

questions / thanks