Game Development with Unity

by Philip Chu

Table of contents

1. Publication Information

2. Overview

Technicat develops games under the Fugu Games and HyperBowl labels using the Unity game engine. This page provides some general information on using this engine, for internal reference, guidelines for development partners, and for anyone else who might find it useful.

Note:

This information is certainly not be up-to-date or complete, or even entirely accurate. For the latest, definitive information, check the Unity web site.

2.1. Platforms

Unity is a multiplatform 3D game engine targeted largely for indie developers and casual games, although the scope appears to be expanding. Platforms include Mac (widget, browser and standalone), Windows (browser and standalone), iPhone and Wii. Note the Mac and Windows version are the same product.

2.2. Games Made

See the Unity games on Reddit, Unity gallery, and Unity forums Showcase and iPhone threads for examples of Unity-made games.

2.3. The Competition

The most similar competitors seem to be Torque and Shiva. See the tools page for other game engines.

2.4. How Much?

Unity is priced at $1500 for the Pro version and now free for the Indie version, which is missing features like 2D image effects, render-to-texture, video, asset streaming, and requires the standard Unity badge and load screen. Unity iPhone is priced similarly for the Pro version and a few hundred dollars for the Basic, but also requires the equivalent desktop version. The Wii version is more expensive, at $15k for indie developers and $30k for a "professional" developer/publisher. All licenses are royalty-free. Compare licenses on the Unity license

chart.

3. Getting Started

3.1. Background

- Visit the Unity site.
- Read the Unity documentation.

3.2. Installation

Download Unity from the Unity download page

3.3. Learn

Go through the official tutorials, sample projects and othe resources.

Check out the many third-party tutorials.

Peruse the Unity wiki for tips and contributed code. That's a great place to contribute your own code.

The Unity Developer Magazine appears to be a well-received, if sporadic, resource.

There aren't as many books on Unity as on Torque and Unreal, but there is Unity Game Development Essentials.

Follow the Unity blog (and check out Unity user blogs).

Note:

3D game development requires knowledge of content creation, programming, game design, and 3D graphics concepts. If you're creating a game yourself and not playing a specialized role in a large team, you can probably get away with being a novice at game design and content creation (make a bad game with stock assets), but you can't get away without knowledge of 3D graphics and programming. Much as I'd like to say Unity is a great way to learn it all, learning it all at once is probably too much to expect. You should read up on the basics, first, before jumping in. See Graphics and Game Development for recommended reading.

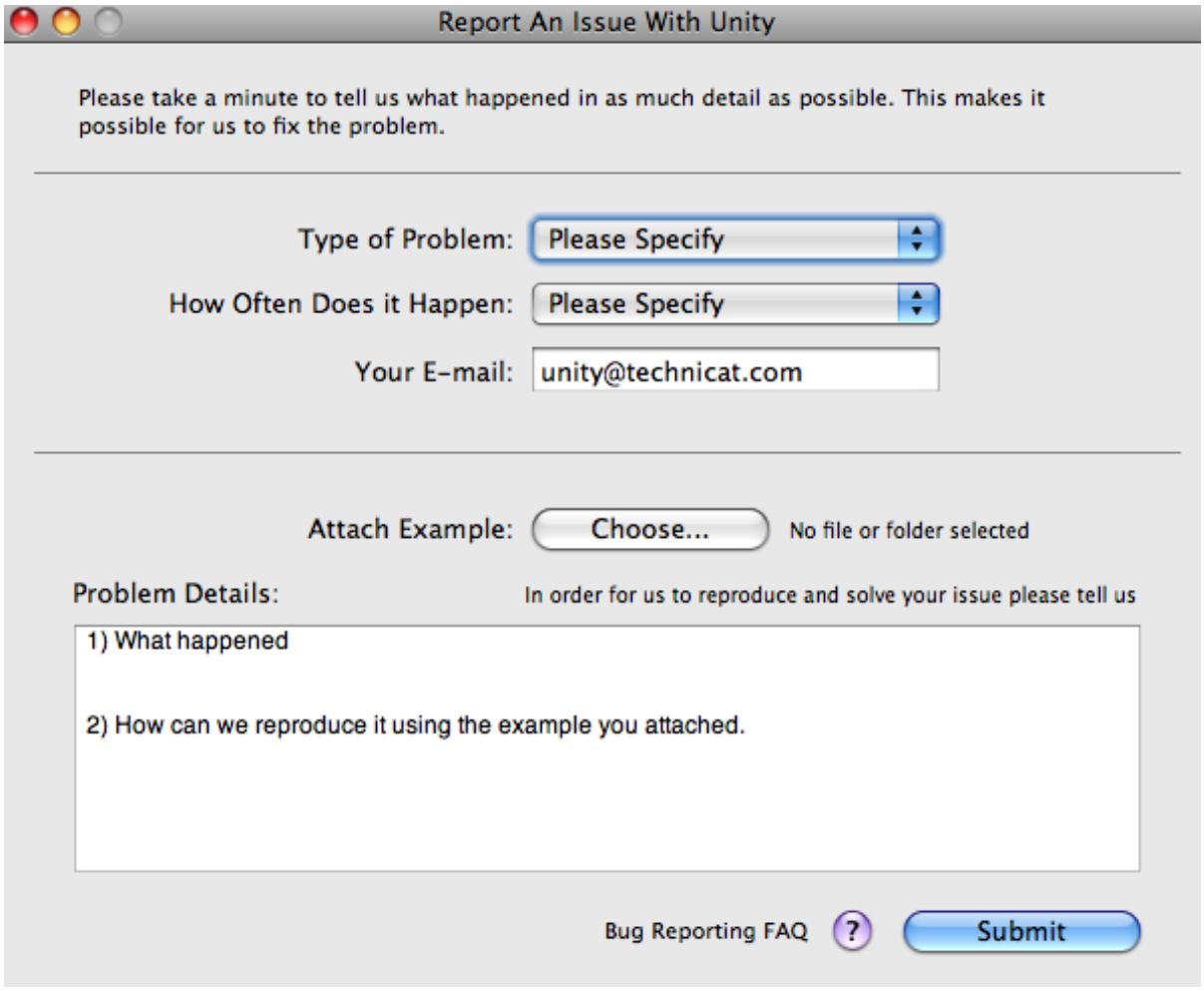
3.4. Asking Questions

- Read the Support FAQ
- Ask specific questions on the Unity answers site.
- Participate in general discussion topics on the Unity forum.

- Check out Unity roadmap
- Explore other Unity-related sites.

3.4.1. Reporting Bugs

If you have a bug that you want fixed, report it via the Bug Reporter (via Help->Report a Bug or find the Bug Reporter app in the Unity applications folder).



See the Unity blog post on effectively reporting bugs. You'll get an automatic email acknowledgment with a link to the new entry in the Unity bug database like this one.

Note:

> The entry is private insofar as no else has the URL until you disclose it, but anyone who has the URL can see all of your bug reports. Removing the last four characters of the URL will leave a link that only displays that specific bug report.

While you're waiting, you can also report the problem and ask for workarounds or commiseration on the appropriate Unity forum, but you'll probably get a reminder that you should report it via the bug reporter.

3.4.2. Requesting Feaures

For a feature request, you have a choice: request it using the bug reporter, vote for it on the Unity feedback page or post it on the Wishlist forum. If you post on the forum or report it via the bug reporter you will likely receive a recommendation to post it on the feedback site, but your votes (and thus posting ability) there are limited. Votes are replenished when a feature you voted for is implemented.

4. Inside Unity

4.1. The World According to Unity

Before talking about how to develop a game, let's discuss what's actually in a game, or at least, a game built with Unity.

4.2. 3D

If you're new to 3D, it's not that a big a deal, conceptually. Everyone (I hope) learned algebra and trigonometry with the Cartesian x,y coordinate system. 3D just requires adding a z-axis, so we specify points as x,y,z.

In Unity, by convention, the y-axis points up. You don't have to stick with this convention, but it makes things easier - otherwise you'll have to change defaults like the direction of gravity, the default orientation of cameras, etc.

Also by convention, one unit in the Unity coordinate system equals one meter in the real world. Theoretically, you could make on Unity distance unit correspond to any real-life distance you want, but all the default distance units in Unity (physics settings, light and camera near/far planes, shadow distances...) assume one Unity unit corresponds to one meter. So in practice, life is a lot easier if you stick with that.

4.3. Scenes

A Unity game consists of one or more scenes. In other game engines, you'd call them levels.

(in fact, some Unity script functions use "level" in their names, but more on that later)

4.4. Game Objects

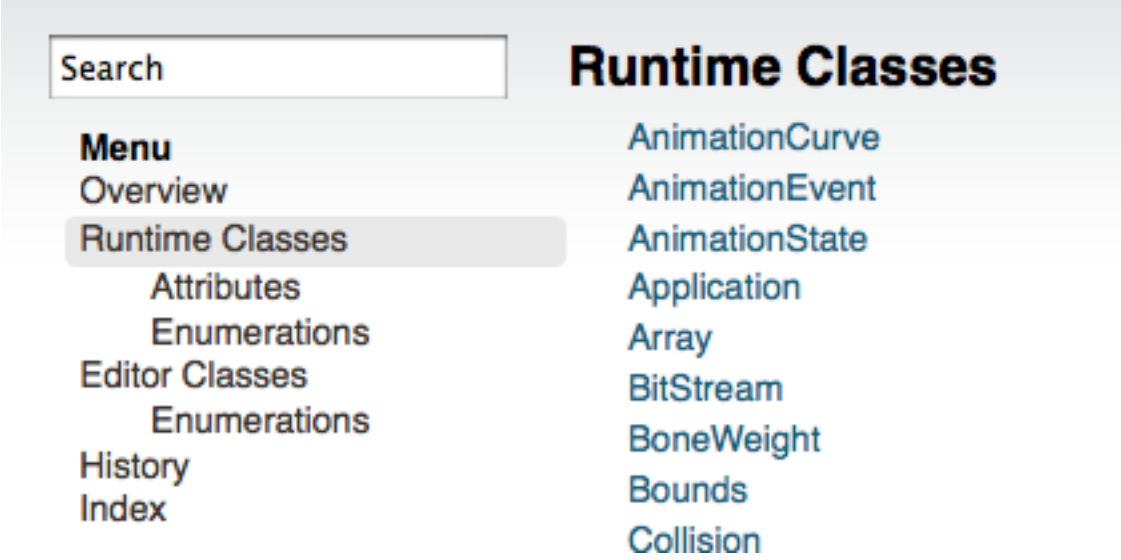
A scene consists of "game objects", often known as "entities" in other game engines. A game object has a name, position and orientation in the scene and other attributes or behavior depending on what type of object it represents.

Game objects can have parent-child relationships amongst each other. The position and orientation of each game object is relative to its parent. This is known in the 3D graphics world as a scene graph, (so the term "scene" make sense). You can sketch out these game object relationships as a graph (or more specifically, a tree, since it's a hierarchy).

Parenting makes sense for game objects that are conceptually grouped together. For example, when you move a car, you want the wheels to automatically move along with the car. So the wheels should be specified as children of the car, offset from the center of the car. When the wheels turn, they turn relative to the car.

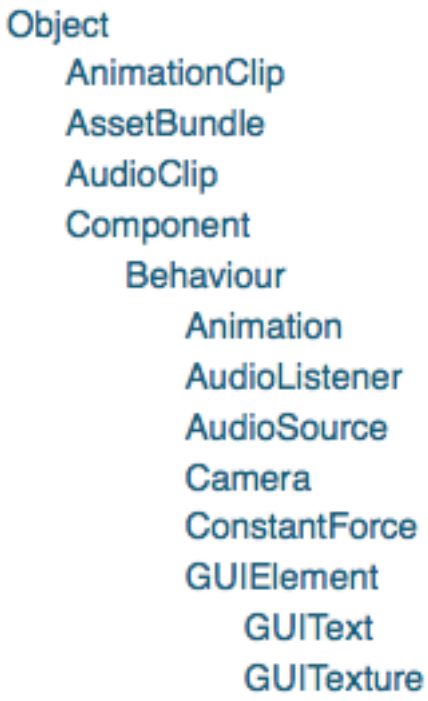
4.5. Classes

Unity is object-oriented. That means each object has a class, and each class can derive from a "base" or "parent" class.



As an object-oriented system, Unity has a class hierarchy. The game object class derives from the object class, as do many other classes. This truncated snippet from the Unity Scripting Reference class hierarchy shows a portion of the class hierarchy descending from

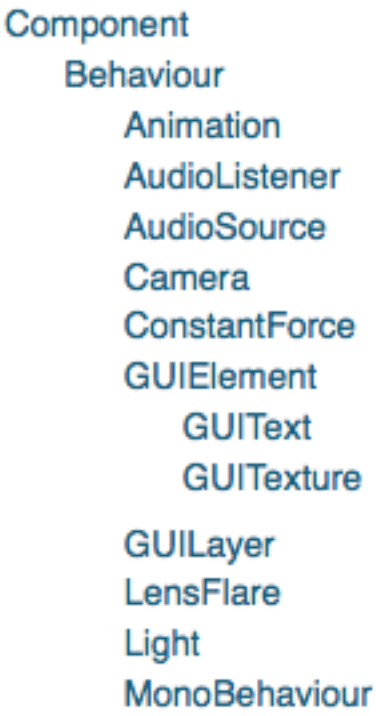
Object.



4.6. Components

Many early 3D systems used inheritance to specialize the behavior of each scene graph node. This could result in some awkwardness in trying to define classes that handled every conceivable behavior you'd want in a single node. As a reaction, "component" sytems have been popular recently. In Unity, for example, each node of the scene graph is of one class, game object, and to specify its behavior, you attach "components" with the behaviors that you want.

Notice in the class hierarchy snippet above that there is a Component class derived from Object. All Unity components are subclasses of Component or subclasses of subclasses of Component. For example, Behaviour (note the British spelling) is a type of Component that can be enabled/disabled. Here's a list of all Behaviours.

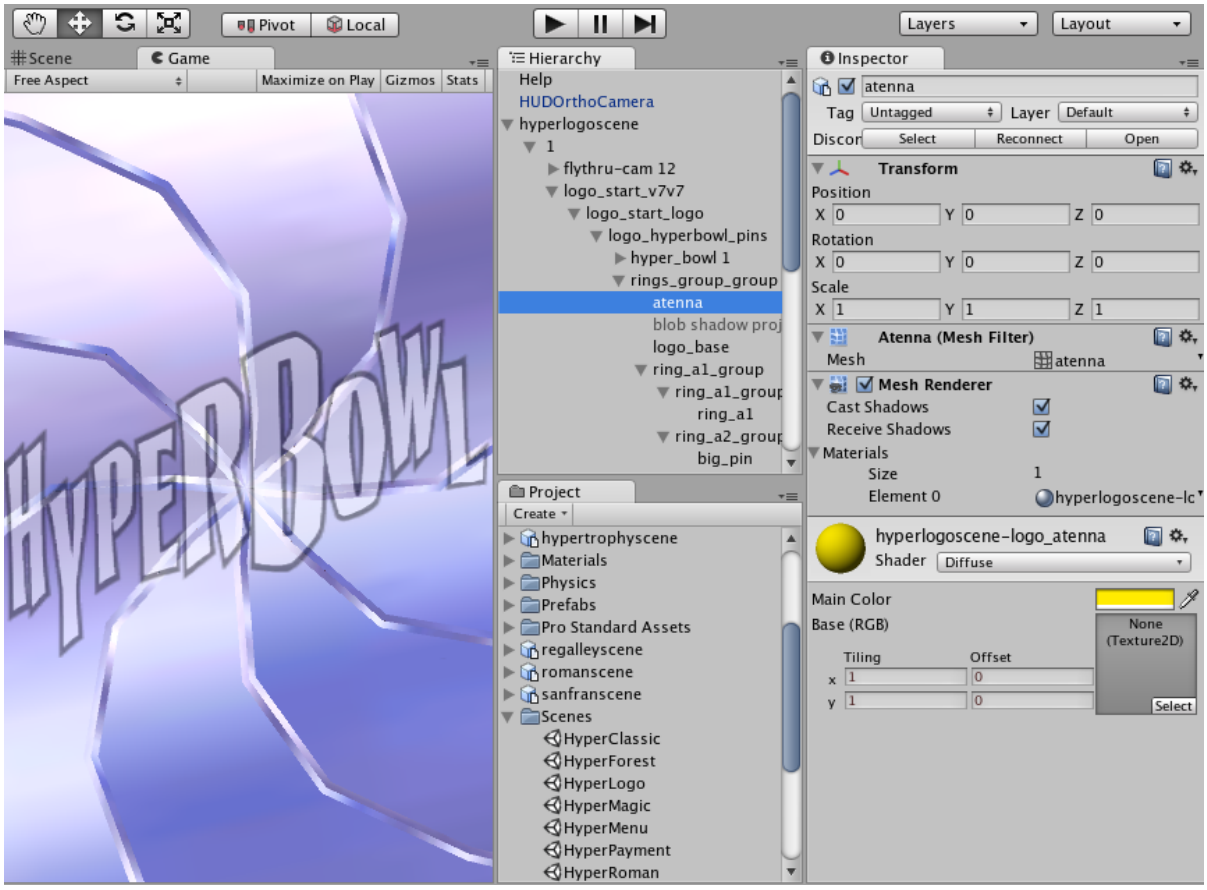


There are many other types of components, too many to list here. The Unity Scripting Reference has the entire list. One advantage of component-based game engines is that they are particularly amenable to drag-and-drop game creation user interfaces. Want to add a light to a particular node in the scene hierarchy? Simply drag a Light component onto that node. You'll see that's exactly how the Unity Editor works. So Unity components are documented not only in the Unity Scripting Reference but also in a Component Reference, and the documentation for each component allows has a link that allows you to switch between documentation for the Scripting usage and the GUI usage in the Unity Editor, which we'll show in the next section.

5. Workflow

5.1. Editor

Now that we know what we're creating, we'll take a quick look at the creation process. Unity games are created as *projects* in the Unity Editor. The Editor displays one Unity *scene* at a time. For example, here is the Unity Editor view of HyperBowl, specifically, the intro logo scene.



Note:

Sometimes when you open an existing project with Unity, it doesn't display a scene even if the project has existing scenes. Don't panic - just double-click the scene you want to to open.

To create a game with Unity, the basic steps are:

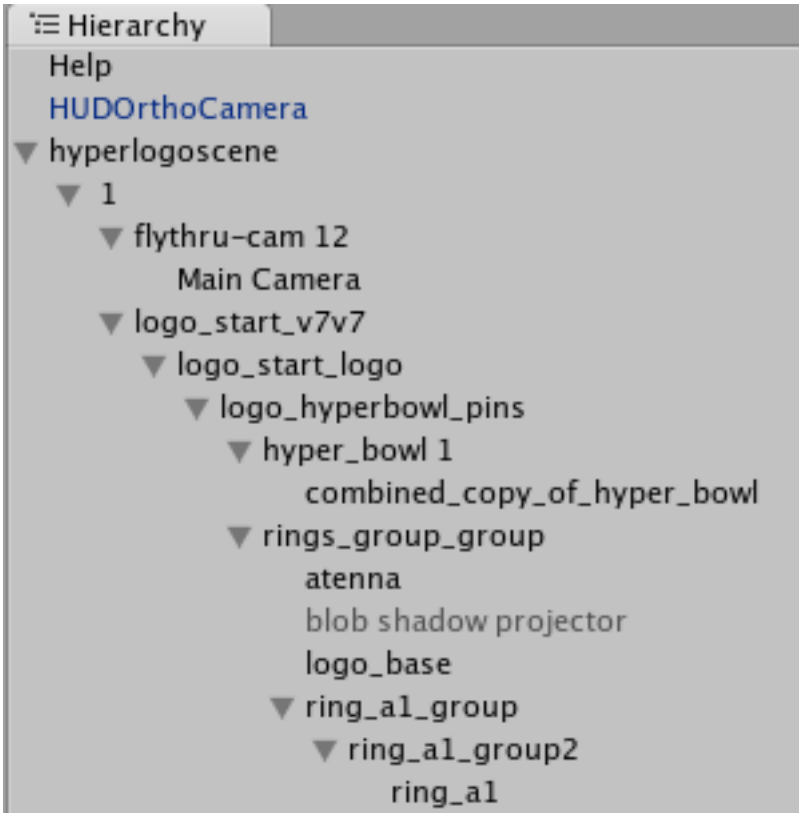
1. Create a Unity project for the game.
2. Import assets.
3. Create a scene for each level.
4. In each scene, select and place assets.
5. Adjust/place the main camera, add new cameras as desired.
6. Add light objects, adjust ambient light.
7. Add/adjust materials to objects.

8. Attach physics materials, colliders, rigidbodies to objects.
9. Write and attch scripts to objects.
10. Test run in the Editor.
11. Publish to the desired platform.

Iteratively, of course.

5.2. Hierarchy

The scene graph for this scene is in the Hierarchy pane.

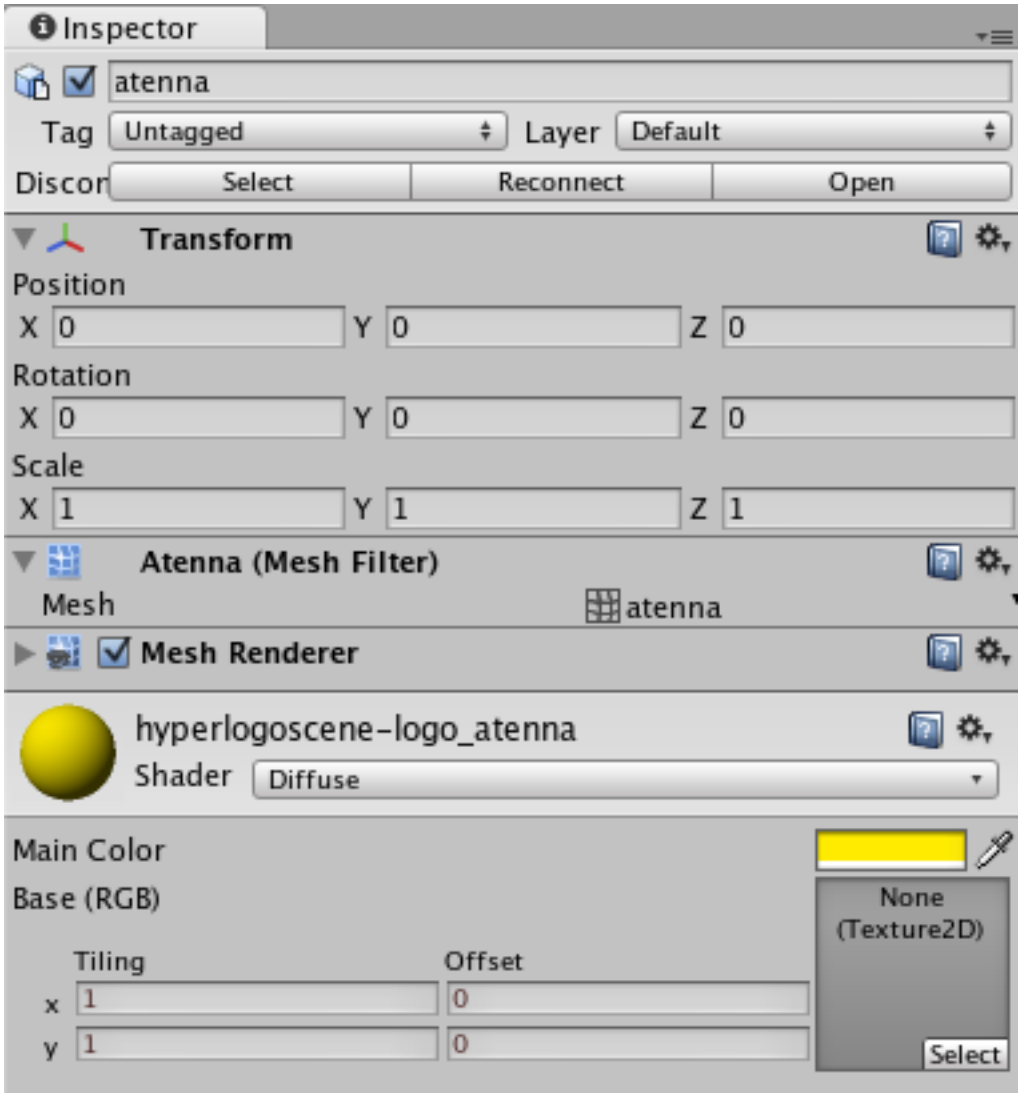


5.3. Inspector

The Inspector pane displays attributes of the game object selected in the Hierarchy pane. This includes the game object name and its components.

We mentioned each game object has a position and orientation. That is stored in the

Transform component. We see the antenna also has a mesh filter component, which contains the actual mesh data, and a mesh renderer, which makes it possible to render the mesh, determines interaction with shadows, and includes materials applied to the mesh)



5.4. Project

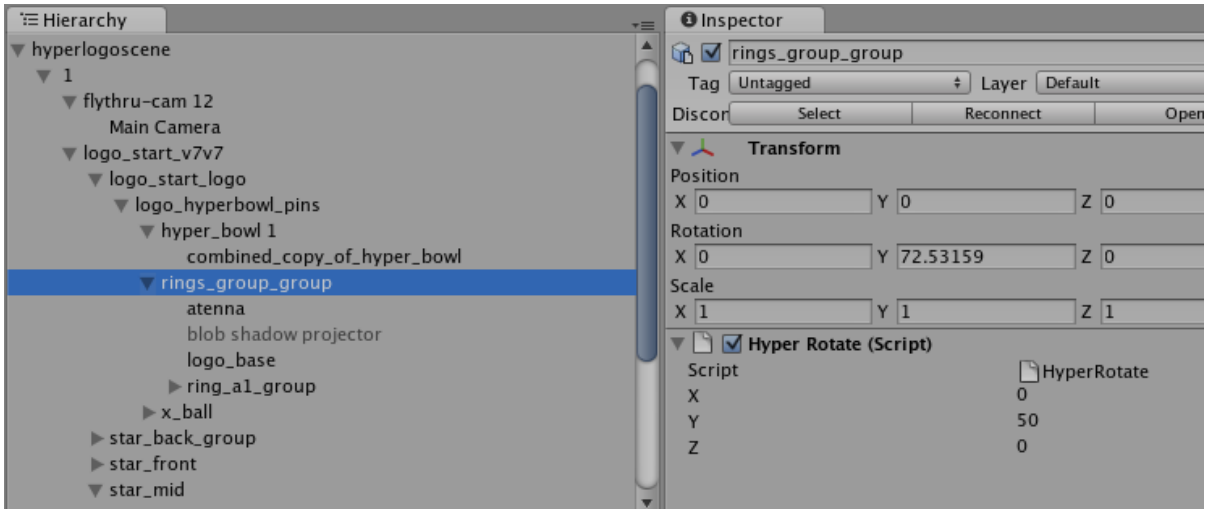
The Project pane lists the available assets, essentially a pool of prototypes you use to construct your Hierarchy. Technically, you could programmatically generate everything in your game (in other words, populate the Hierarchy) programmatically, but typically you'll

drag assets from the Project pane into your Hierarchy pane and then modify the copy in the Hierarchy pane as appopriate. In other words, the Project pane shows what you can use to construct your game and the Hierarchy pane displays what is actually in the game.

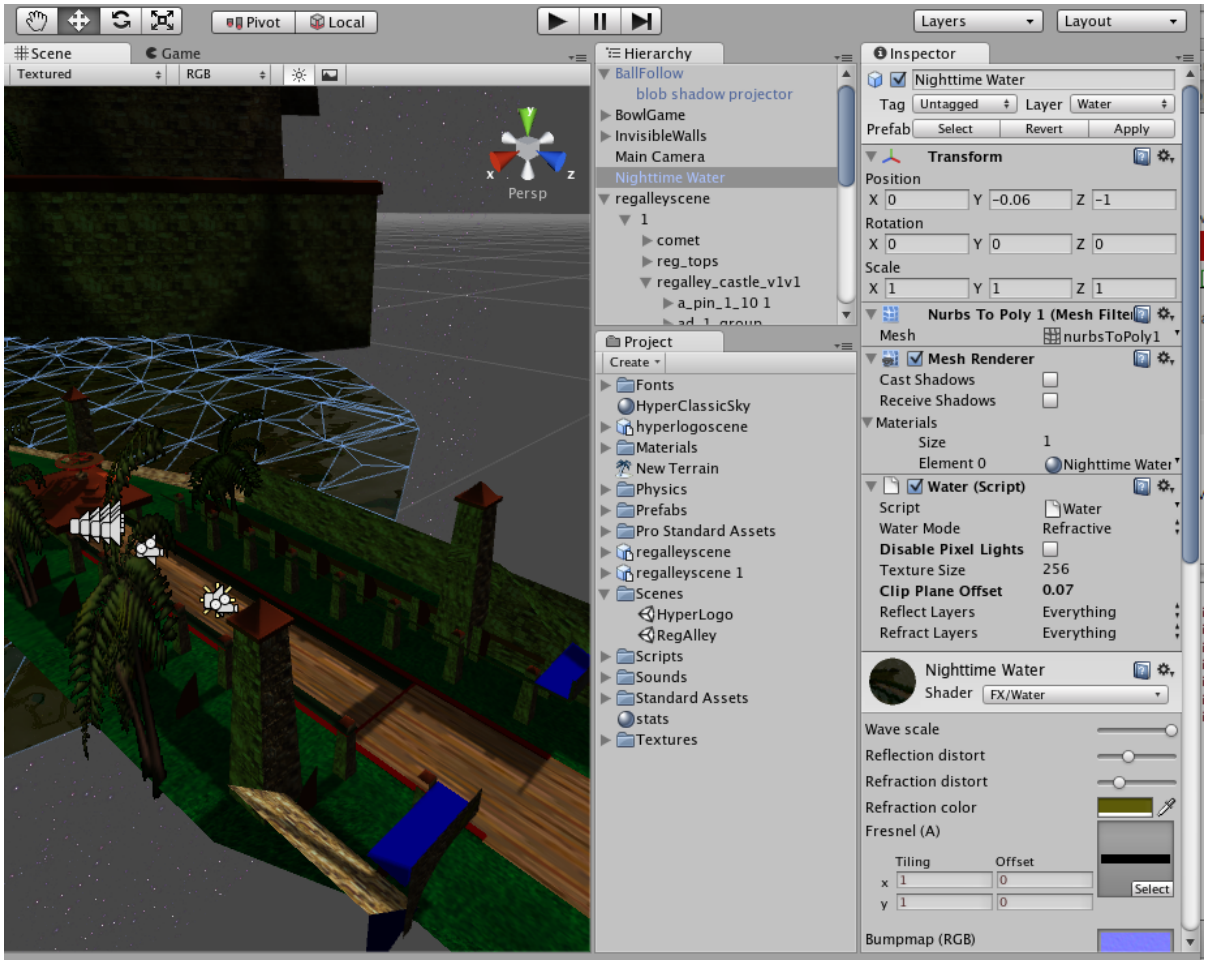
5.5. Preview

To test it, just hit the Play button.

Scripts are also attached to game objects as components. Here a script is attached to the parent rings node solely in order to rotate all the rings. When we hit Play, we'll see the rings rotate and also see the rotation values in the ring transform update in the Inspector. Note how all the objects in the hierarchy beneath the parent rings node rotate along with it, but the globe, which is outside that hierarchy, doesn't



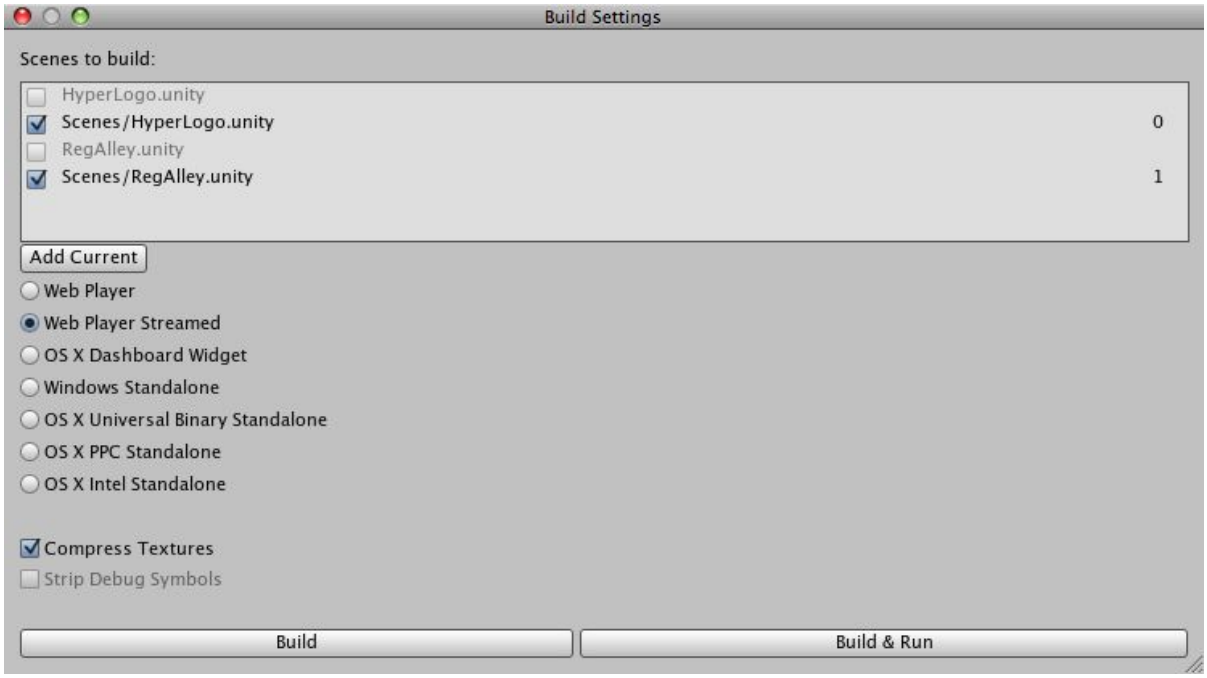
A game will typically have more than one scene, e.g. multiple levels with a scene for each level, and maybe an introductory scene like the HyperBowl logo scene above. Here's the scene for the HyperBowl Classic lane.



The logo scene has a script to transition from the logo scene to the bowling scene, called HyperStream, as we are taking advantage of the scene streaming feature available for web builds. The script is attached to a 2D text object so we can conveniently display the streaming progress and a "hit enter to continue" message.

5.6. Publish

Finally, we can publish the game as a web player. As noted above, we're using the streaming feature available in web builds (so while the logo is spinning, the next level is loading).

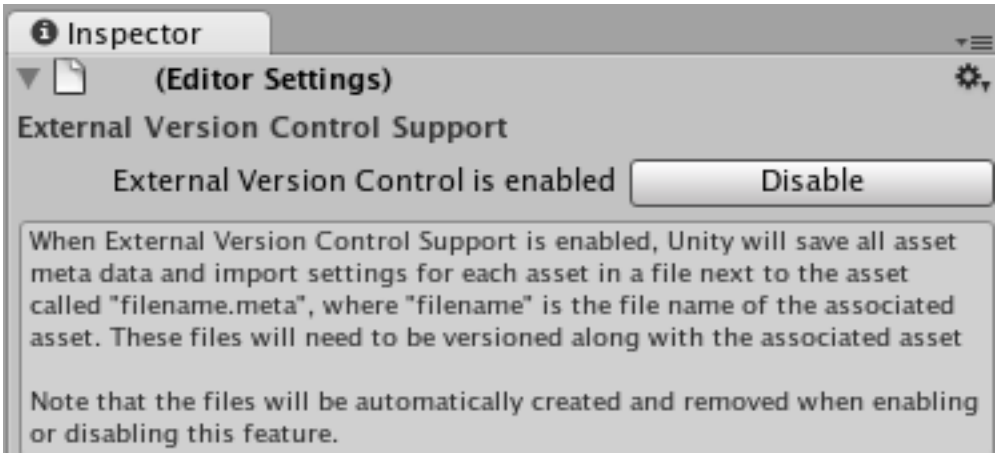


6. Collaboration

6.1. Version Control

The Unity Editor has integrated support for the Unity Asset Server to provide version control and workgroup collaboration for Unity projects.

The metadata in the project Library directory, which tranks the relationships among the project Assets, doesn't play nicely with other source/version control systems. Unity provides some minimal support for those systems by optionally maintaining a metadata file for each Asset file. This option can be enabled in the Editor settings.



To start using an external version control system with Unity, follow the using external version control instructions

Note:

The external source/version control support just provides compatibility with those systems, not integration. You'll have to perform updates and commits with the external system outside of the Unity Editor.

6.2. Distributing Projects

Even if you're creating content to hand off to a Unity developer and not using any collaboration tools, I highly recommend you import your assets into Unity and deliver them in the form of a Unity project folder or Unity package file. This allows you to ensure the assets will appear in Unity as intended, with the desired materials, shading, lighting, colliders, particle behavior...as much control as you want to exert over the final product. The gneral workflow workflow might be:

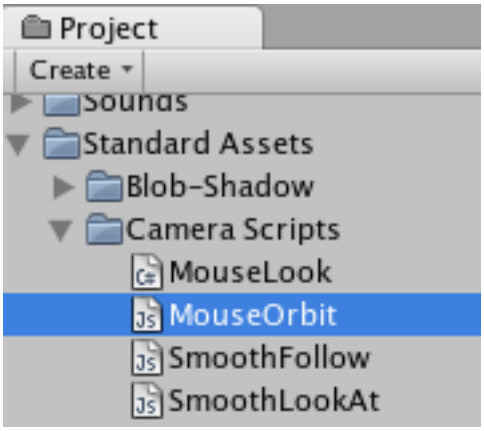
1. Import assets into a Unity project.
2. Set up controls/camera.
3. Set up the appropriate materials with desired textures and shaders.
4. Any defects in the assets, reimport, repeat.
5. When satisfied, deliver the entire Unity project, or export as a .unitypackage file

6.3. Model Preview

Finalizing the artwork in Unity allows you to inspect and verify your work and also maintain the most control over the final look - otherwise you'll have a programmer deciding what

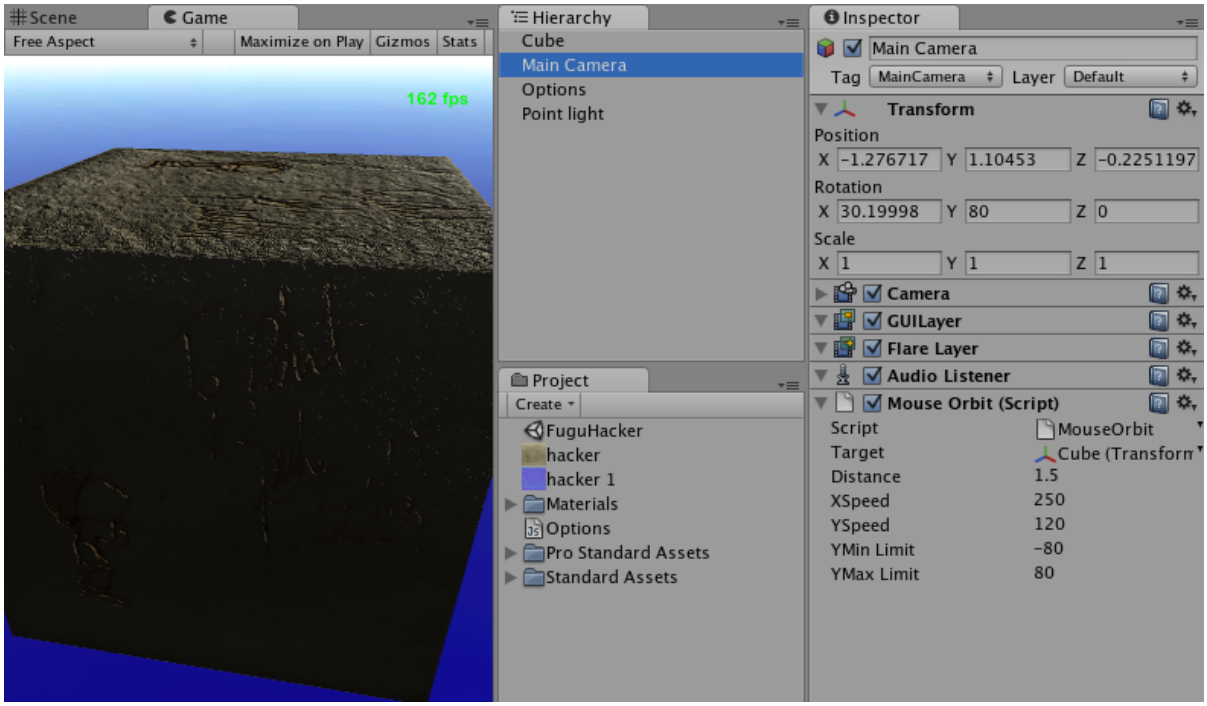
textures and material parameters to use. If you're providing textures for skyboxes and particle effects, which need to be constructed and viewed within Unity, then it's even more important for the artist to complete the work within Unity so there's no confusion over how the assets are to be used.

Previewing a model in Unity is simple, using the provided mouse orbit camera script.



1. Start Unity and create a new project (you can reuse the same project over and over if you're just using it to inspect assets
2. Import your asset as described above
3. Drag the asset into the scene
4. Drag the StandardAssets/Scripts/Camera/MouseOrbit script to the main camera in the scene
5. Select the camera, and then drag the asset in the scene into the "target" slot of the camera script.
6. Hit Play to mouse-orbit the camera around the asset. Adjust the camera script "distance" attribute of the camera script to move the camera closer or farther

Here's a project in which I just wanted to see if a badly neglected sketch given to me in high school by Berkeley Breathed would make a nice bump map (yeah, I meant to age and crumple it!). I scanned the sketch, created a Unity project, imported the jpeg of the sketch as a texture, created a cube, created a bump material with the sketch, added the mouse orbit script to the camera, and voila!



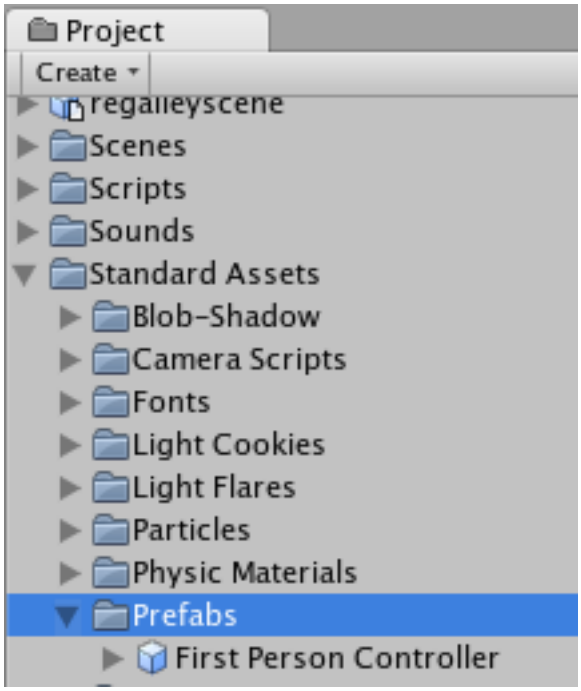
6.4. Animation Preview

6.5. Audio Preview

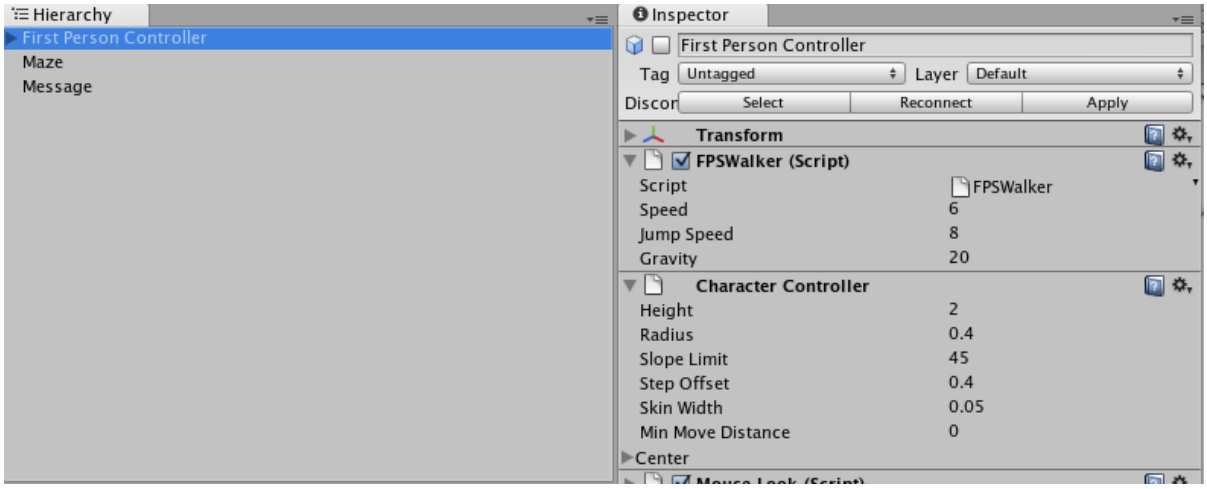
6.6. First-Person Scene Preview

Artists and designers responsible for creating environments need to work within Unity even more. Besides placing and testing imported assets, the developer will probably need to create/place/test cameras, lights, terrain, skyboxes, physics settings, all of which can only be performed in the Unity Editor.

Testing an environment for a first-person game is nearly as simple as previewing a single model. Instead of attaching a camera script to the camera, you just drag a first-person controller into the scene.



1. Start Unity and create a new project (you can reuse the same project over and over if you're just using it to inspect assets
2. Import your assets as described above
3. Drag the asset into the scene
4. Drag the StandardAssets/Prefabs/FirstPersonController prefab into the scene where you want the first-person character to start.
5. Hit Play



6.7. Game Design

If you're specifying a game design, I cannot stress enough how important it is to sketch it out as a state diagram. It seems to be a highly under-taught device, but it is invaluable. If you can specify a state diagram of your game, then you can visualize and communicate to others how the game will operate. Once it is in the form of a state diagram, then it is ready to code (see the scripting section below). Conversely, if you cannot specify it as a state diagram, then you don't have a game design - you have a vague idea of a game.

Example here

Some cases may seem so simple you don't need one. Fugu Maze, for example, doesn't have an explicit state machine - you're in the maze, you reach the end, rescrable and start over. But already, that's a few states:

maze state machine

The existing Fugu Maze code would be a lot simpler if it was running a single game state machine. Lesson - always start with a state diagram and implement with a state machine. Perhaps you'll just have one state, but probably not.

6.8. Packaging

- Projects
- Packages

7. Assets

7.1. Importing

Assets are imported into Unity simply by placing them in the project Assets directory. You can also use the *Asset->Import Asset* menu item. An automatic import then takes place and the asset then shows up in the Unity editor project display.

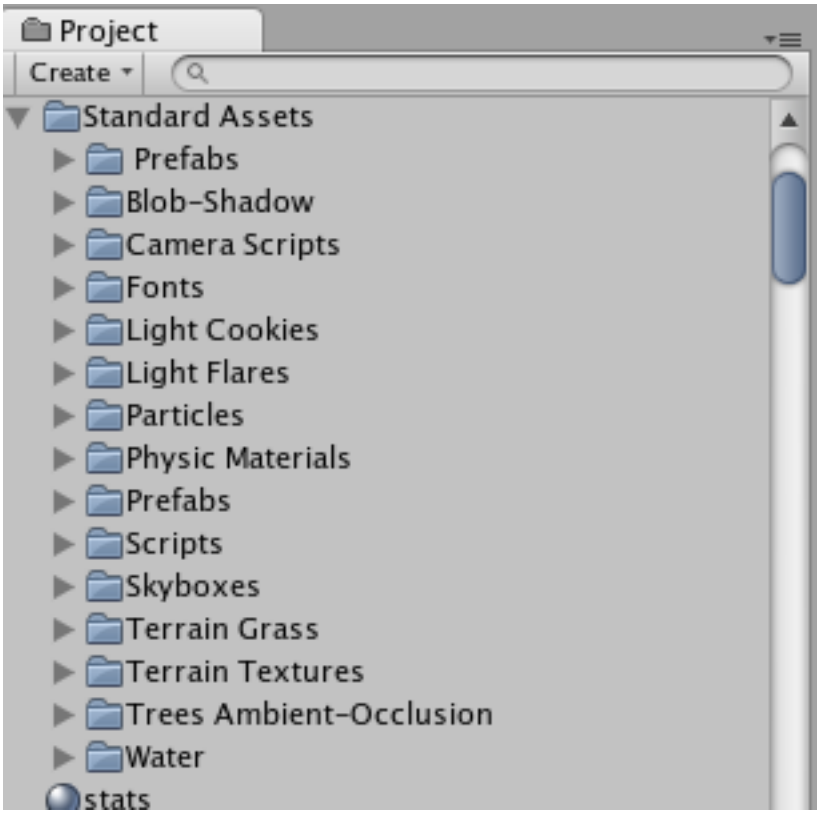
You can then adjust the import settings (there's no way to do it beforehand) and then hit the reimport button.

Note:

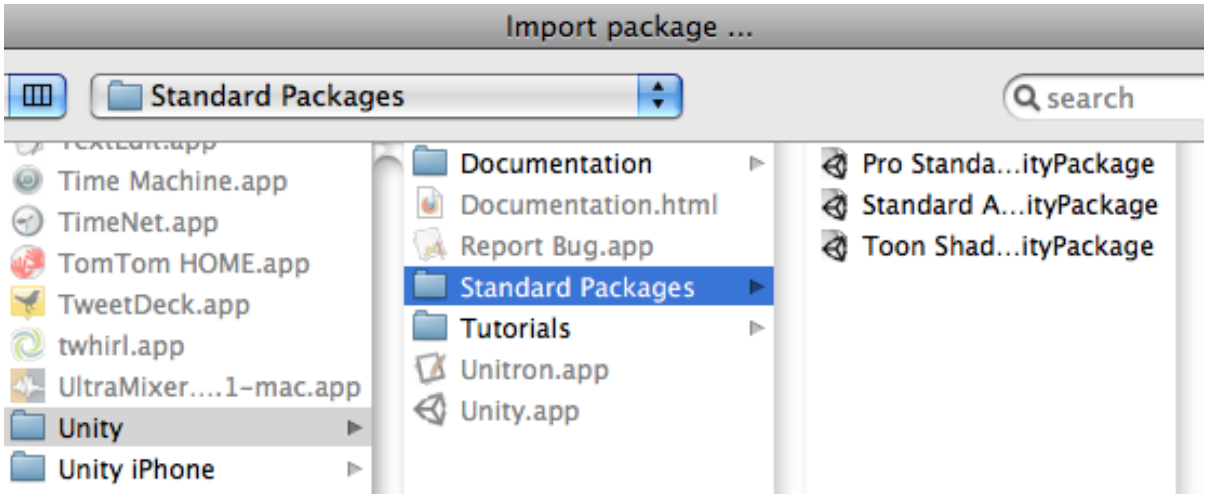
One of the less helpful defaults in Unity is the Scale Factor used in import. The sharp-eyed user may notice the default is 0.01, for legacy reasons. So adjust the Scale Factor and reimport as necessary.

7.2. Standard Assets

When you create a new project, you'll be prompted whether to immediately import Standard Assets. If you're new to Unity, you definitely should include these, at least to reference as examples. Don't worry about bloating your game - unused assets don't contribute (much) to the final game size.



If you need import or reimport Standard Assets later, (e.g. if you modify elements of Standard Assets and want to revert to the original), you can find the corresponding .unitypackage in the Unity installation.

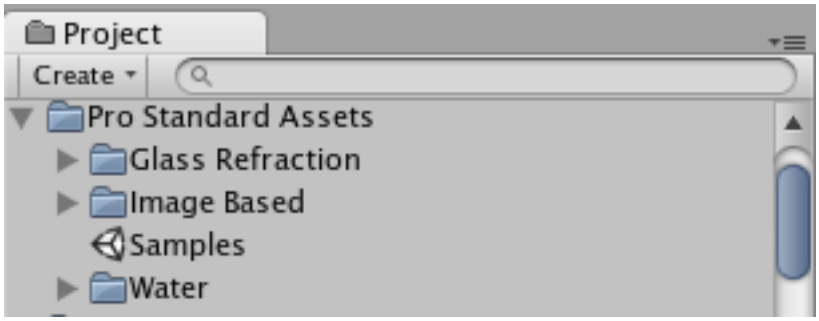


Note:

When you upgrade to new versions of Unity, be sure to also reimport the standard assets, so you get the latest stuff (and fixes).

7.3. Pro Assets

If you have Unity Pro, you also have the option of including the Pro Standard Assets.



7.4. Organizing Assets

Note:

Once an asset has been imported, you can move or delete it in the Project pane, but don't move or delete it from within the Finder. Unity tracks relationships among the assets in a project, so moving or deleting those assets outside Unity will mess up your project.

7.5. Animation

Animation is imported as FBX (or Collada?) files. If character animations are set up and imported according to one of the conventions for Importing Character Animation then the resulting character contain a list of the named animations that can be controlled by the script.

Note that animations tend to take a lot of memory. If an animation is simple, the like the HyperBowl logo rotation, it's better to let a script handle it.

7.6. Meshes

Officially, several formats are supported, (see Meshes (User Manual) but the most reliable is FBX. OBJ and Collada support appear to be implemented by using those importers in the FBX SDK, so they are only as worthy as the FBX SDK support. For example, OBJ files created by PolyTrans hang during import and also when run through the Autodesk FBX

converter, so no surprise. Watch out, the default conversion scale during import is 0.01 for legacy support reasons.

7.7. References

- Wiki tips and tricks
- Working with Assets (User Manual)

8. Scripting

8.1. Languages

The scripting system is based on Mono, an open-source version of .NET. Like .NET, mono supports many programming languages, but Unity only supports C#, Boo, and Javascript. Each language has it's pros and cons. C# has the advantage of having an official definition and many instructional books and online tutorials, is the closest thing to a main language on .NET and Mono, and thus used in a lot of commercial and open-source software. But C# is the most verbose choice.

Boo seems to be well-liked by Python programmers but is probably used by the smallest number of Unity users and correspondingly has the least support.

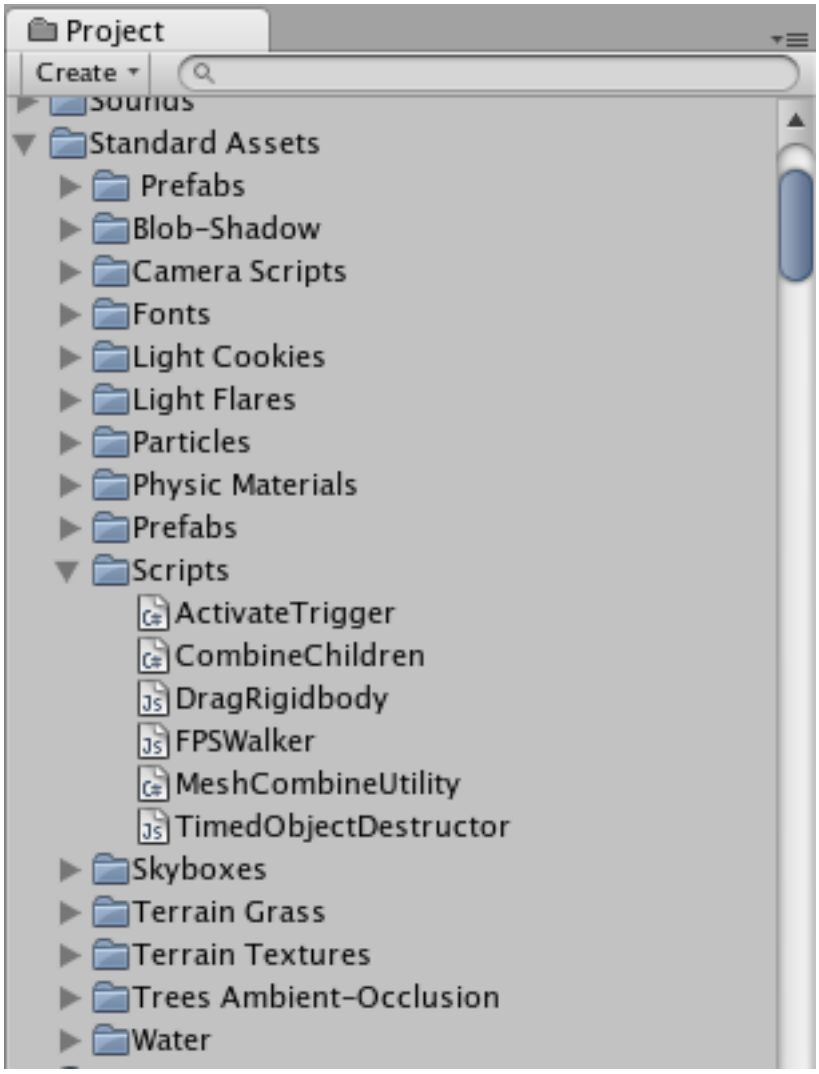
The version of Javascript in Unity is more succinct but apparently derived from the Boo implementation and not quite standard Javascript (so it's been suggested that it be called UnityScript). Most of the scripts supplied with Standard Assets are in Javascript, although a few are in C#. I mainly use Javascript, with the exception of some C# code from open-source libraries, so I'm only going to show Javascript examples here.

Note:

While it is possible to combine scripts written in different languages in the same Unity project, it is tricky. See Script Reference section on Script Compilation. Sticking with one language makes things a lot easier.

8.2. Scripts are Assets

Scripts are assets just like textures and models. Several scripts are provided in the Standard Assets.

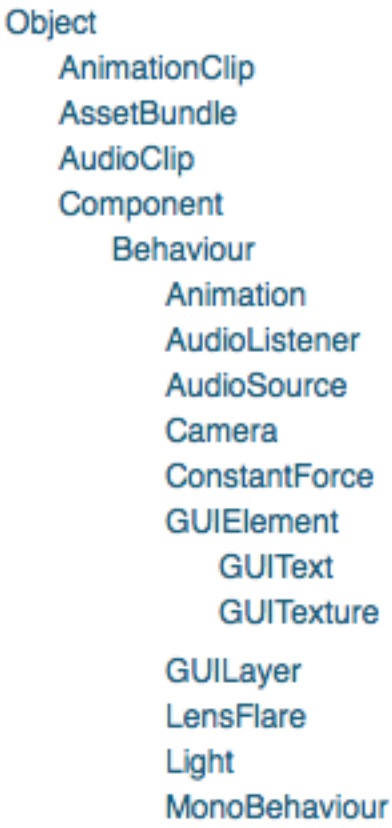


Scripts can be imported like other assets (via the Assets-Import menu item or dragged into the Assets folder).

8.3. Scripts Are Components

Unity is programmed with scripts - there is no C++ programming involved (unless you write plugins or buy a source license). Scripting in Unity is a bit different from traditional programming, but similar to the use of Linden Scripting Language in Second Life and Lua in CryEngine - individual scripts are attached to objects in the game. This lends itself to an object-oriented design where you write particular scripts to control particular game entities.

In Unity, scripts are components and are attached to game objects just like other components. The Unity Manual section on using scripts details how to create scripts and attach them to objects. Specifically, scripts are of the MonoBehaviour class, a direct subclass of Behaviour, which as we noted before, is a Component that can be enabled/disabled.



Note:

The term "object" is often loosely bandied about, in the case of Unity typically referring to a game object or a game object plus its components. But remember a script component is a distinct object from its parent game object.

8.4. Scripts are Classes

Each script actually defines a new subclass of MonoBehaviour. With a Javascript script, Unity treats this definition as implicit - the new class is named according to the file name, and the contents of the script are treated as if they occur within a surrounding class definition. In C# scripts, the class definition is explicit. `

MonoBehaviour is actually subclass of Behaviour which, as we noted before, is a Component that can be enabled/disabled.

Note:

Be careful that you don't define new classes with the same name as existing .NET classes. Normally, you would define a namespace to avoid collision with existing names, but Unity doesn't currently support namespaces. In my FuguType typing game, I created a class called Dictionary to contain all the words in the game. Unfortunately for me, there is a .NET Dictionary class. The game would run for a while, but eventually it would crash.

8.5. Script Structure

A bunch of code attached to an object isn't very interesting if none of it executes. Scripts add behavior to objects by implementing callback functions that are invoked by the Unity engine during the lifetime of a game.

Awake is called when the object is loaded.

Start is called after Awake but only when or if the object is active (and only after the first activation, not repeatedly if the object active status is switched on and off and on again).

Update is called once per frame while the object is active, after Start.

FixedUpdate is called after every fixed update interval (the same interval used for physics updates)

Sometimes you want to perform actions when a game object is activated or deactivated. For example, the HyperBowl scoreboard plays an animation every time it's activated. If we placed that code in the Start function, it would only run at most once. Instead, we use OnEnable

And there is a corresponding OnDisable

Other callbacks are invoked under certain event, like physics.

- OnCollisionEnter
- OnCollisionStay
- OnCollisionLeave

You don't need to specify every single one of them - just the ones you need. See the Unity Scripting Reference for the full explanation of each.

8.6. Reaching Out

Even with callbacks, scripts wouldn't be useful if they couldn't at least control their parent game objects. A script can refer to its parent game object via its gameObject member variable. A script can call GetComponent to access any component of the game object, but a number of member variables act as shortcuts to common components.

Note:

There is some additional performance expense in using the shortcut variables, it's best to assign the result to a script variable if you want to refer to the component repeatedly, particularly in OnUpdate.

8.7. Active and Enabled

We've been playing fast and loose with the term "active". When we say a script callback is called when a game object is active, we really mean the game object is active and the script is enabled (hence the naming of the OnEnable and OnDisable callbacks.

Aside from the callbacks, we can examine and set the game object's active member variable and the script's enable member variable.

8.8. Coroutines

Unity operations aren't thread safe, so Unity scripts support coroutines. See the Coroutines overview.

8.9. State Machines

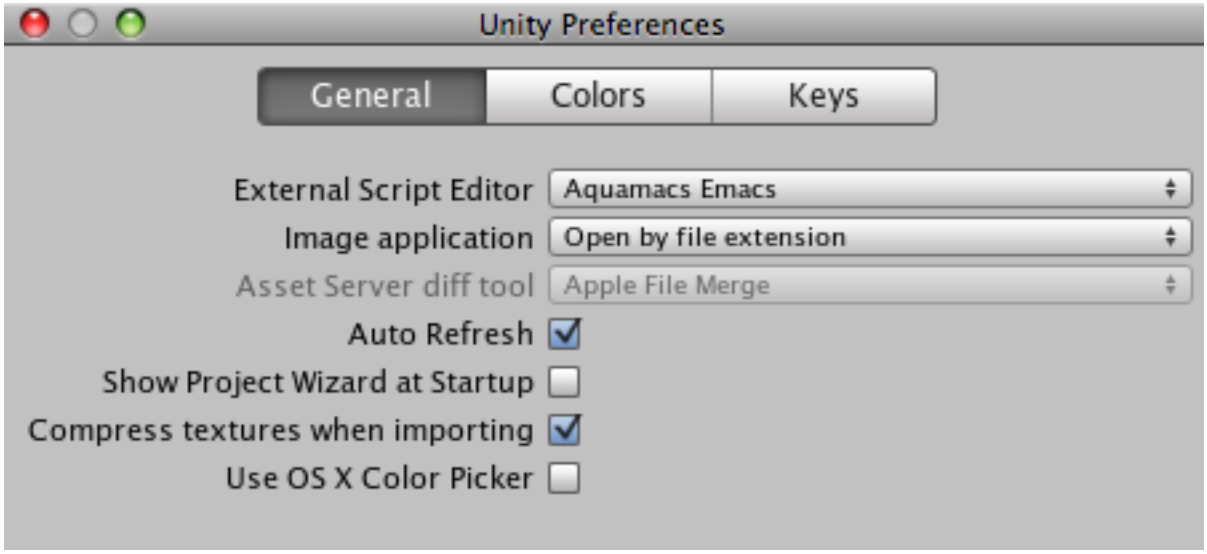
Coroutines provide an easy way to implement finite state machines. FSM's are an excellent way to to define the behavior of an object, so much so that some scripting systems explicitly support them (e.g. Second Life and CryEngine).

8.9.1. State Machine

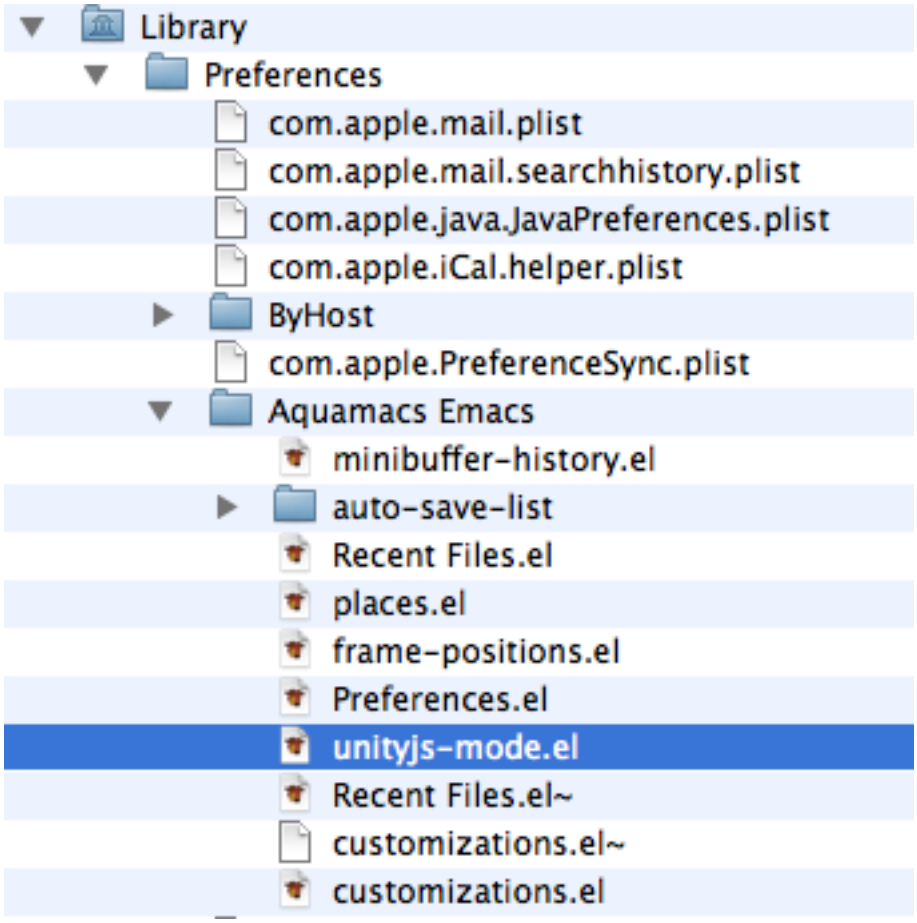
```
function Start ()
{
state = "BowlInit";
while (true) {
yield;
yield StartCoroutine(state);
}
}
```

8.10. Editing Scripts

When you double-click on a script in the Project window or click on Edit for a script displayed in the Inspector, Unity brings up a script editor. Unity has a default script editor called Unitron, but you can specify a different editor in the Preferences. For example, I use a Mac version of Emacs called Aquamacs.



The Javascript mode that comes with Unity doesn't sport the nice Unity-function color-coding of Unitron, but the Unity staff has provided a similar Emacs Unity-Javascript mode



8.11. Third-Party Libraries and Tools

The Unify community wiki features many large and small scripts.

UnitySteer is an open source Unity steering package patterned after OpenSteer

Ninja Portal offers the Unity Dialog Engine.

Angry Ant offers pathfinding and behaviour tree libraries.

FeedReader is an RSS reader.

8.12. References

See the Script Reference Overview

9. Camera

9.1. Camera Component

In 3D graphics, the viewpoint is represented as a *camera*. When you create a new Unity scene, it always starts out with a "main" camera.

9.2. Main Camera

Specifically, a camera is a game object with a camera component attached. (see the Camera Component Reference)

9.3. Layers

Any number of cameras can be added to a scene and any number can be active at a time. That sounds like it could be messy, but you can assign layers to the objects in the scene and specify the layers visible to each camera. You might use this to implement a HUD camera, for example - specify all HUD game objects to be in a HUD layer and the HUD camera to only see the HUD layer.

9.4. Split-Screen

Another reason you might have multiple cameras active simultaneously is for a split-screen, e.g in a co-op multiplayer game.

9.5. Perspective Cameras

Most cameras are "perspective cameras", which display objects with perpective foreshortening, i.e. they look smaller at a distance.

9.6. Orthographic Cameras

Orthographic cameras have no perspecive foreshortening. You might use an orthographic camera for a HUD camera. Or for a 2D or isometric game.

9.7. Camera Control

Camera control is an underappreciated aspect of game design and programming. Camera control usually just involves the same translation and rotation operations you can perform on any game object, but you can also access any of the camera component properties via script

(see the Camera Script Reference.

The Standard Assets include a basic set of comera control scripts, but it's worth reading up on the subject, e.g. the comprehensive book Real-Time Cameras, and some excellent articles in the Game Programming Gems series.

10. Physics

10.1. Overview

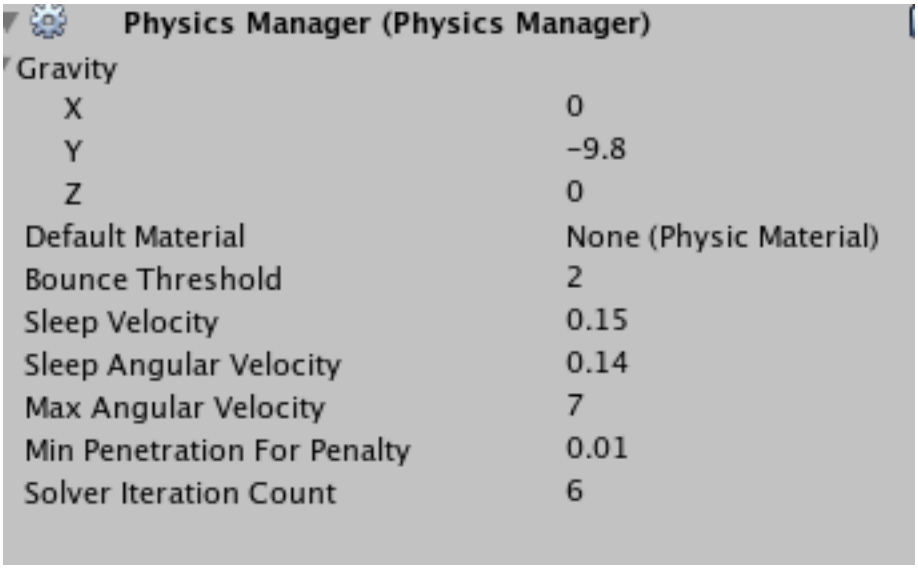
Unity physics is based on a version of the PhysX physics simulation engine. It is not current with the latest nVidia releases but it's worth downloading the free Windows SDK and reading the documentation to understand it's model and capabilities.

Note:

Don't try to implement your own physics - PhysX is very sophisticated and it's unlikely you'll produce something as good. Check out the engines and references in Tools if you don't believe me.

10.2. Global Parameters

Global PhysX properties are settable in the Unity Editor via the Physics Manager.



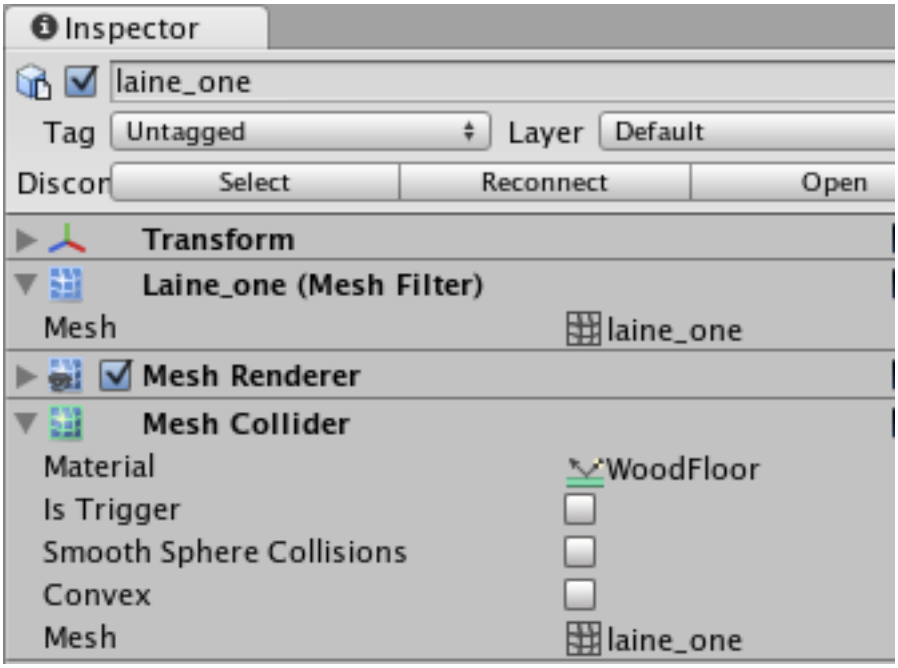
Anyone who remembers high school physics should recognize the gravity setting. Since the y-axis points up in Unity (vs. the z-axis in some applications and engines), the y gravity

value is 9.8 meters/second^2. Notice the default values assume a meter corresponds to one Unity distance unit.

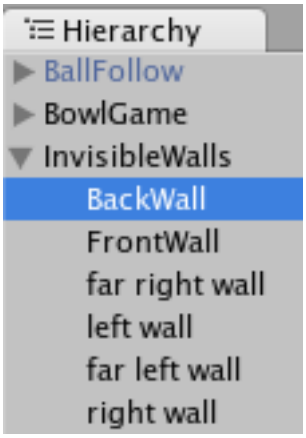
10.3. Colliders

Collisions are detected among collision *shapes*, which in Unity are known as colliders. Colliders are added to a game object as components.

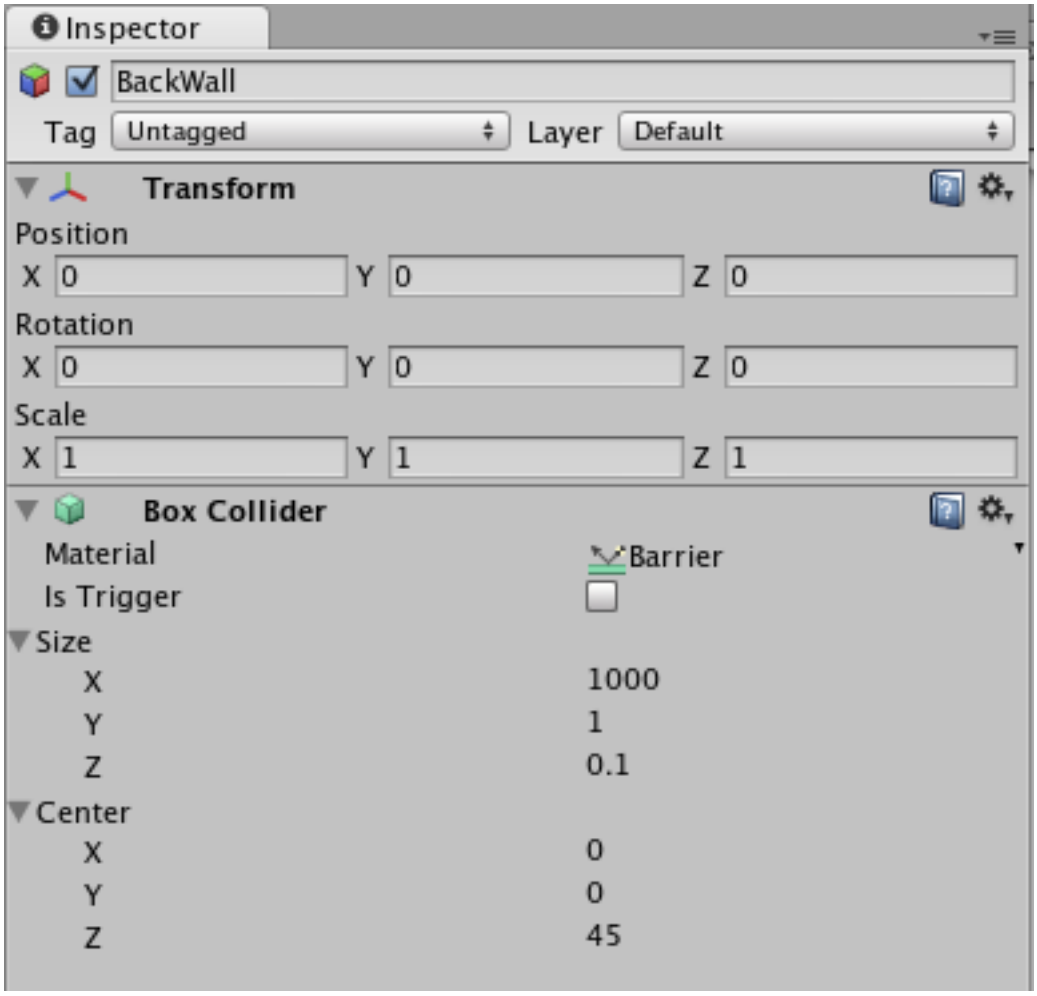
Often, you'll want a collision shape to follow the surface of arbitrary static geometry. That's what a mesh collider is for. The HyperBowl Classic lane does have some non-flat areas on the ground, so it has a mesh collider that uses the mesh in the same game object as its source.



But *primitive colliders* are much more efficient, so use them instead whenever possible (and of course, if an object is not going to collide with anything, save the physics engine some checking and don't have a collider at all). For example, HyperBowl has a number of "walls" that just use plane colliders.



Each wall is a game object with a plane collider attached. Most are placed in alignment with visible walls, but the back wall is never in view, so there is no geometry there.



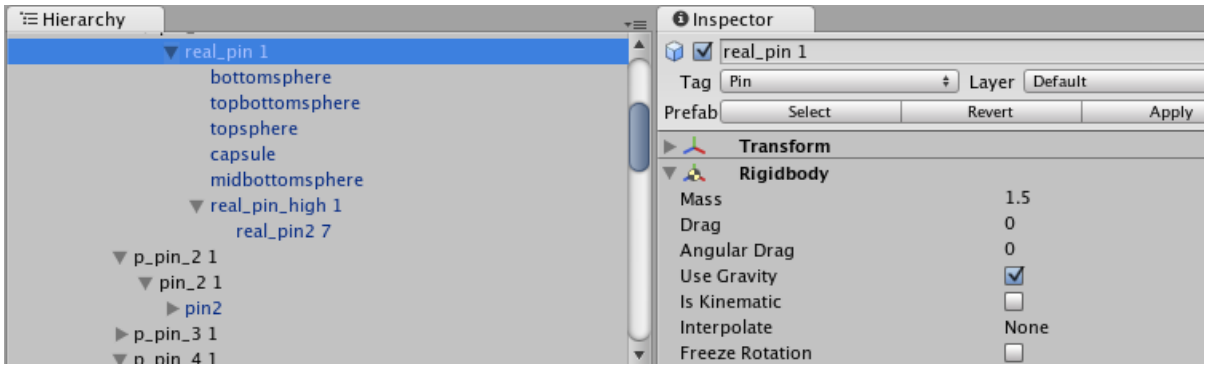
Note:

When you add a primitive collider to a game object with a mesh, Unity initially sizes the collider to encompass the mesh. This is a great convenience and also a neat way to find out the the extents of your object.

10.4. Compound Colliders

Sometimes a single primitive collider is perfect (like a sphere collider for a ball) or a good enough approximation, but what if you have a dynamic object that's not perfectly round or flat or in the shape of a cube or capsule? You can combine multiple colliders under one game object to approximate a more complex collision shape. The HyperBowl bowling pin is obviously such a case - for that we still have one rigidbody but use several sphere colliders of

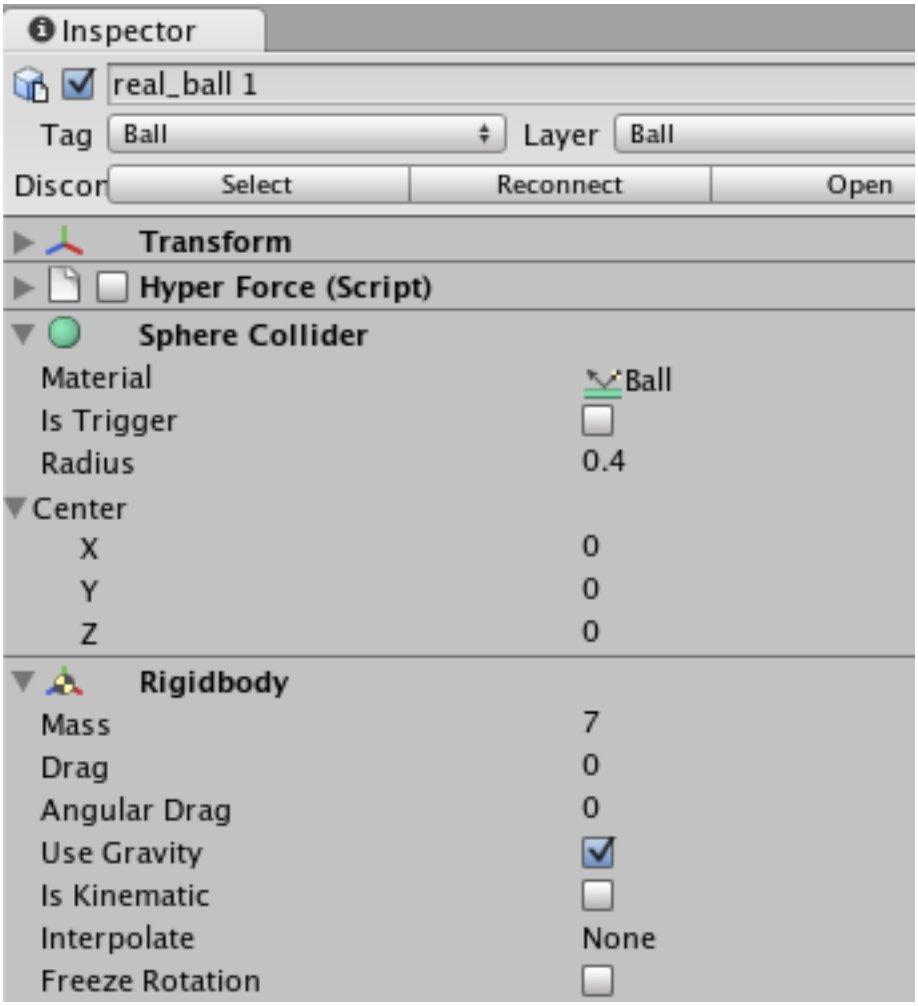
different radii, plus a capsule along the neck.



(show gizmos)

10.5. Rigid Bodies

Collision detection is useful but collision reaction makes things interesting. We employ rigid-body simulation (as opposed to soft-body simulation of cloth, hair, or soft-body parts like, er, potbellies) by, well, adding rigid bodies to game objects with colliders. A game object with a rigid-body component will react to collisions and other forces, including gravity. For example, we certainly want the HyperBowl bowling ball to collide with the walls and bowling pins, roll along the ground and follow the law of gravity.



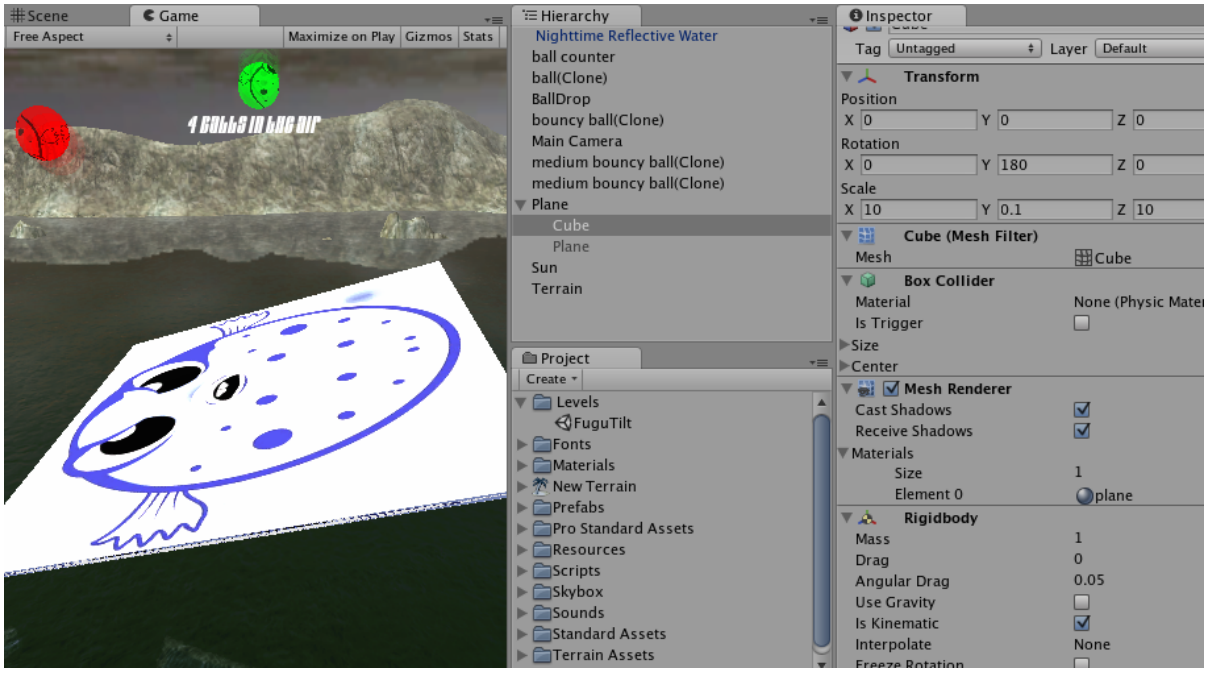
Note:

For good performance and robust collision detection, always try to use primitive colliders for rigidbodies. If you have to use a mesh collider, set it to convex.

10.6. Kinematic Bodies

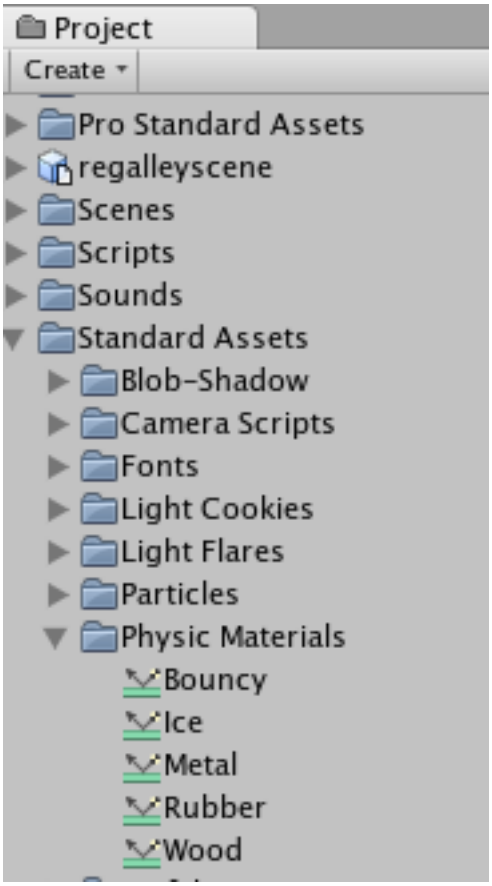
Any moving collider needs a rigidbody, even if you don't want it subject to forces. If it's going to move by animation or code, set the rigidbody to *kinematic*. For example, my Fugu Tilt game features balls dropping on a mouse-controlled plane. The balls have sphere colliders and rigidbodies responding to gravity - the plane has a box collider and rigidbody

marked as kinematic.

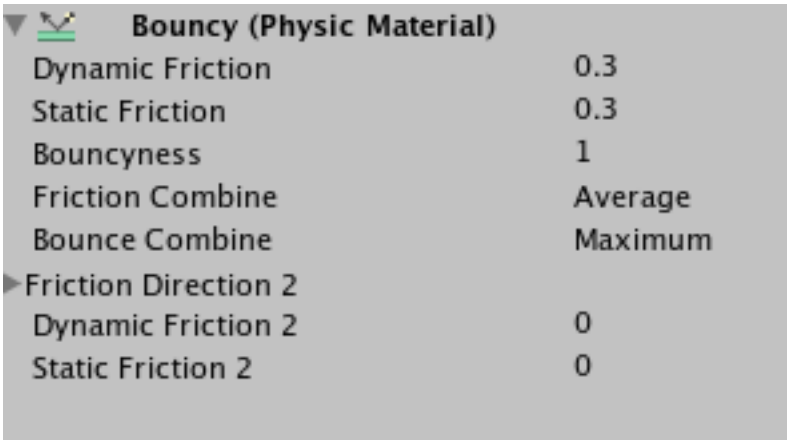


10.7. Physics Materials

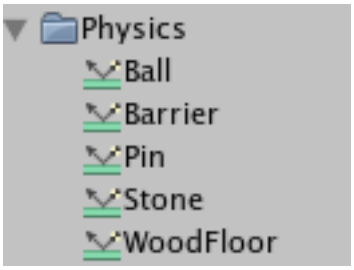
Collision reactions among colliders are determined partly by their surface properties, which in Unity are encapsulated as *physics materials*. The Unity Standard Assets includes a set of representative physics materials.



If you select the bouncy material, for example, you can see the material has properties such as bounciness and friction, each ranging from 0 to 1.



For very simple examples, you can use and tweak the standard physics materials, but in a typical game, you will create a physics material for each distinct type of surface (often corresponding to render materials, hence the reuse of the term "material"), HyperBowl has these physics materials:



10.8. Intersections

Colliders aren't just for collisions. Any time you want to calculate intersections, colliders can help. The callbacks OnMouseEnter, et. al. work on objects with colliders. The Physics raycast and check functions operate on colliders.

10.9. Triggers

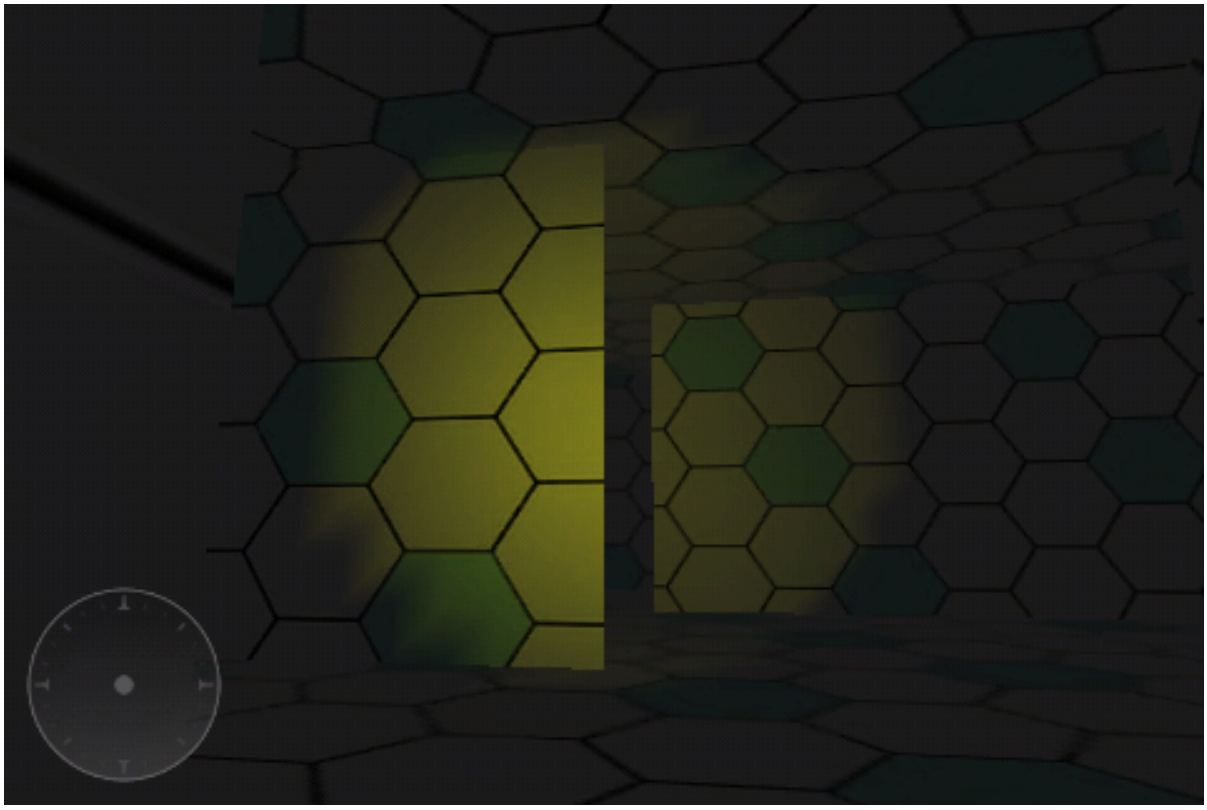
See the Unity Reference Manual.

11. GUI

There are a variety of ways to implement a user interface or HUD display in Unity.

11.1. GUIText and GUITexture

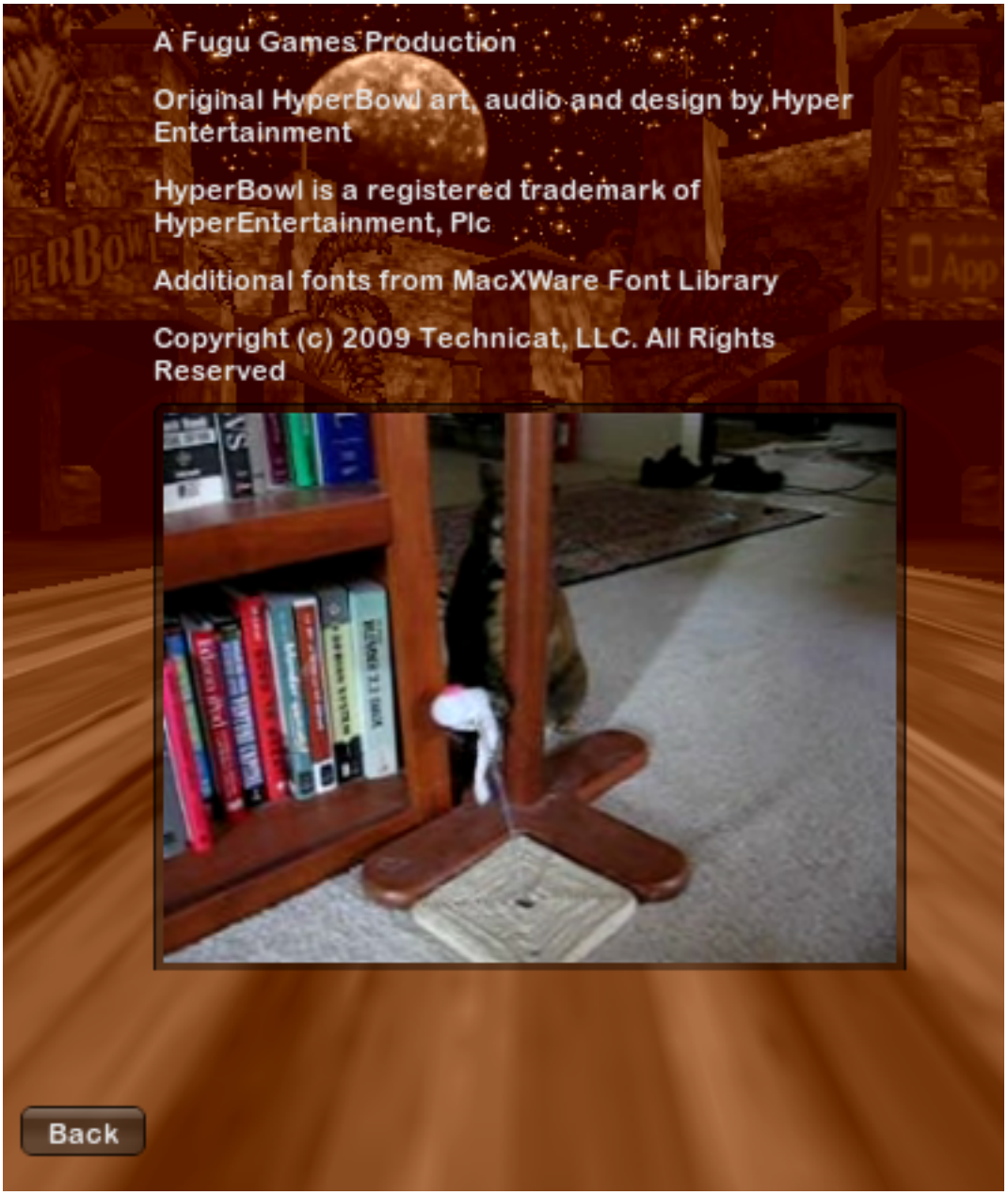
Unity provides GUIText and GUITexture for 2D text and textures, respectively, that are positioned on the screen in screen coordinates. For example, Fugu Maze on the iPhone has a GUITexture representing a move-forward button on the lower-left portion of the screen.



GUITextures respond to OnMouse events, so they can conveniently implement buttons.

11.2. UnityGUI

UnityGUI is a full-fledged GUI framework that supports buttons, checkboxes, sliders, layouts, and custom skins. I use UnityGUI to implement a pause menu (available on the Unify wiki) in all my games. Here's a page from the menu that uses the UnityGUI to display credit text, a movie texture, and a button to exit.



11.3. 3D GUI

You can of course create a 3D GUI, and it's actually quite easy to create one independent of the rest of the 3D scene, since you can have multiple cameras and layers. For example, if

your HUD consisted of 3D objects, you might place all of them in a HUD layer and add a camera that only views the HUD layer.

Typically, a GUI/HUD camera would be orthographic, but it doesn't have to be. HyperBowl has two HUD cameras, an orthographic one that displays the elements at top and a perspective one for the scoreboard, since the animated score numbers use perspective in zooming up to the camera and back into the scoreboard.

12. Networking

Unity implements LAN-level multiplayer using RAKnet.

SmartFox and Exit Games offer Unity libraries that work with their multiplayser servers. Jumpstart uses the Unity web player in conjuncton with SmartFoxServer.

See the Networked Multiplayer section of the Unity manual. Also check multiplayer game resources and tools.

13. Browser

13.1. The Plugin

A distictive feature of Unity is the ability to run in a web browser, using the Unity brower plugin. The 3D browser plugin market has never been really established (remember VRML?), although the apparently defunct Shockwave player seemed to have a good run. Now Unity seems to have a good a claim on the market lead as anyone else.

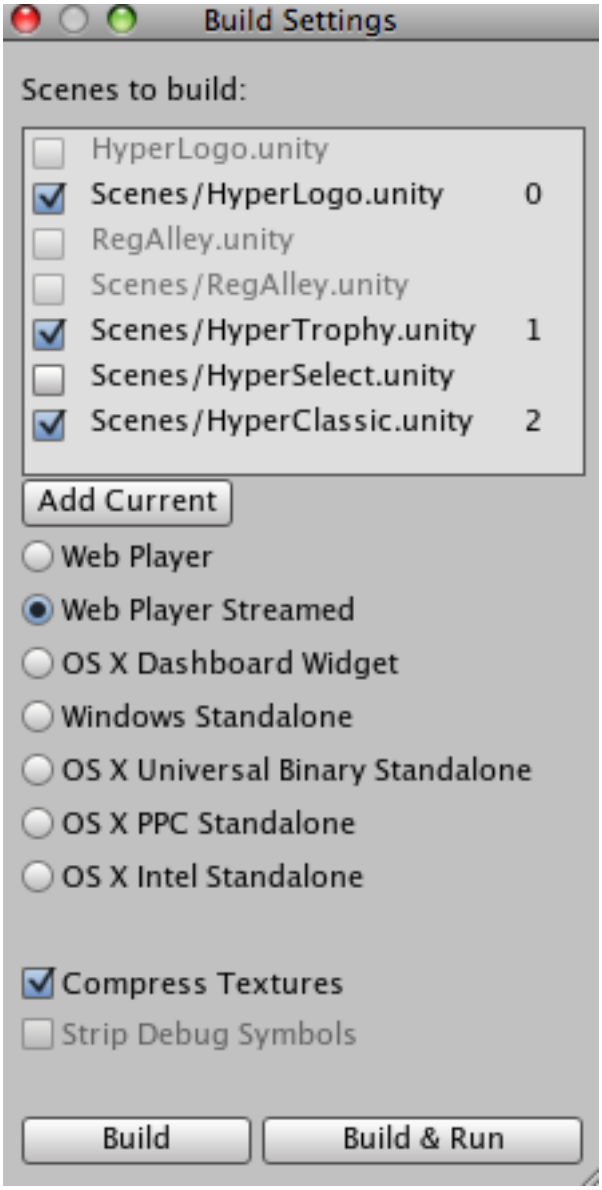
When you publish a web player, Unity will generate some containing HTML which you can customize for deployment.

Note:

Don't mix Unity 1.x files (ending with .unityweb) with Unity 2.x builds (ending with .unity3d). One type cannot load the other (reliably, at least) and note there is different HTML for embedding each.

13.2. Streaming

When you publish a web build, you have the option to create a streaming web player.



If you have more than one scene, you almost certainly want to use streaming. Unity web players published with streaming have access to the Application streaming functions.

13.3. Loading Another Web Player

One web player can load another by calling WWW.LoadUnityWeb This can be convenient if

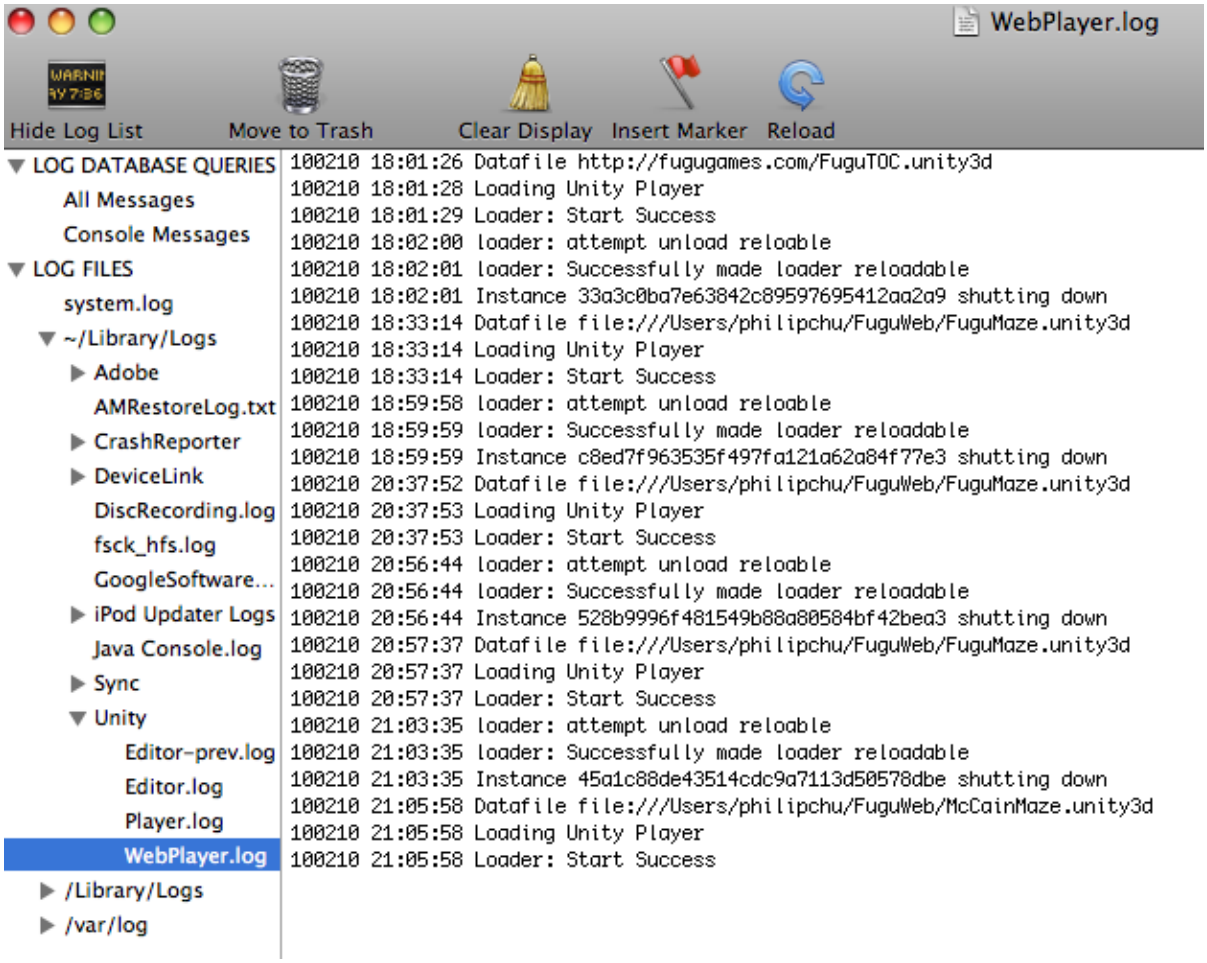
you have a number of separate web players that for one reason or another you don't want to glom together. For example, on Fugu Games I have a simple startup web player that lets you select among a collection of web players.

13.4. Limitations

- Only runs on MacOSX and Windows. Not Linux.
- Application.ScreenCapture not supported
- Application.OpenURL will load the new page over the page containing the web player (there is no "window" target as in Flash Acionscript's getURL)

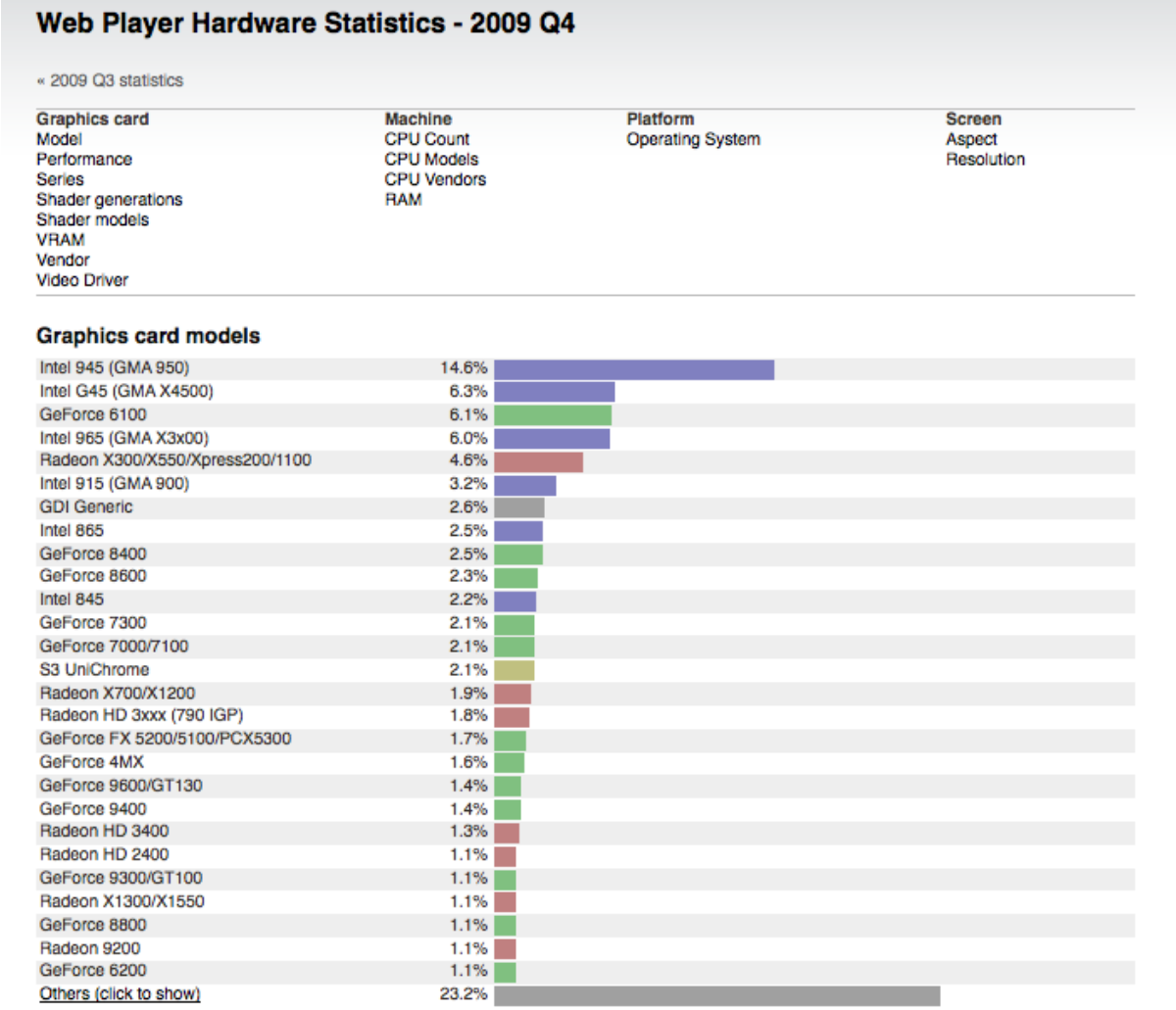
13.5. Debugging

Once a web player is running, you can still debug it by checking the log. (This should be the first thing you do if you see your web player freeze or crash). The Unity documentation on Web Player Debugging explains where to find the log on various platforms.



While developing, you should disable the Unity plugin cache by visiting the Unity setup page (same page linked to by the "Setup" option in the plugin context menu). Otherwise you may wonder why you're not seeing the change you just made. Also, if you always have to wait for your player to load, you won't forget about minimizing the download time.

With PC games you have to worry about differences among OS versions and hardware, especially graphics cards. But with browser games you have all that in addition to varying web browsers. Take a look at the Unity web player statistics.



Web Player Hardware Statistics - 2009 Q4

« 2009 Q3 statistics

Graphics card	Machine	Platform	Screen
Model	CPU Count	Operating System	Aspect
Performance	CPU Models		Resolution
Series	CPU Vendors		
Shader generations	RAM		
Shader models			
VRAM			
Vendor			
Video Driver			

Graphics card models

Intel 945 (GMA 950)	14.6%
Intel G45 (GMA X4500)	6.3%
GeForce 6100	6.1%
Intel 965 (GMA X3x00)	6.0%
Radeon X300/X550/Xpress200/1100	4.6%
Intel 915 (GMA 900)	3.2%
GDI Generic	2.6%
Intel 865	2.5%
GeForce 8400	2.5%
GeForce 8600	2.3%
Intel 845	2.2%
GeForce 7300	2.1%
GeForce 7000/7100	2.1%
S3 UniChrome	2.1%
Radeon X700/X1200	1.9%
Radeon HD 3xxx (790 IGP)	1.8%
GeForce FX 5200/5100/PCX5300	1.7%
GeForce 4MX	1.6%
GeForce 9600/GT130	1.4%
GeForce 9400	1.4%
Radeon HD 3400	1.3%
Radeon HD 2400	1.1%
GeForce 9300/GT100	1.1%
Radeon X1300/X1550	1.1%
GeForce 8800	1.1%
Radeon 9200	1.1%
GeForce 6200	1.1%
Others (click to show)	23.2%

Note:

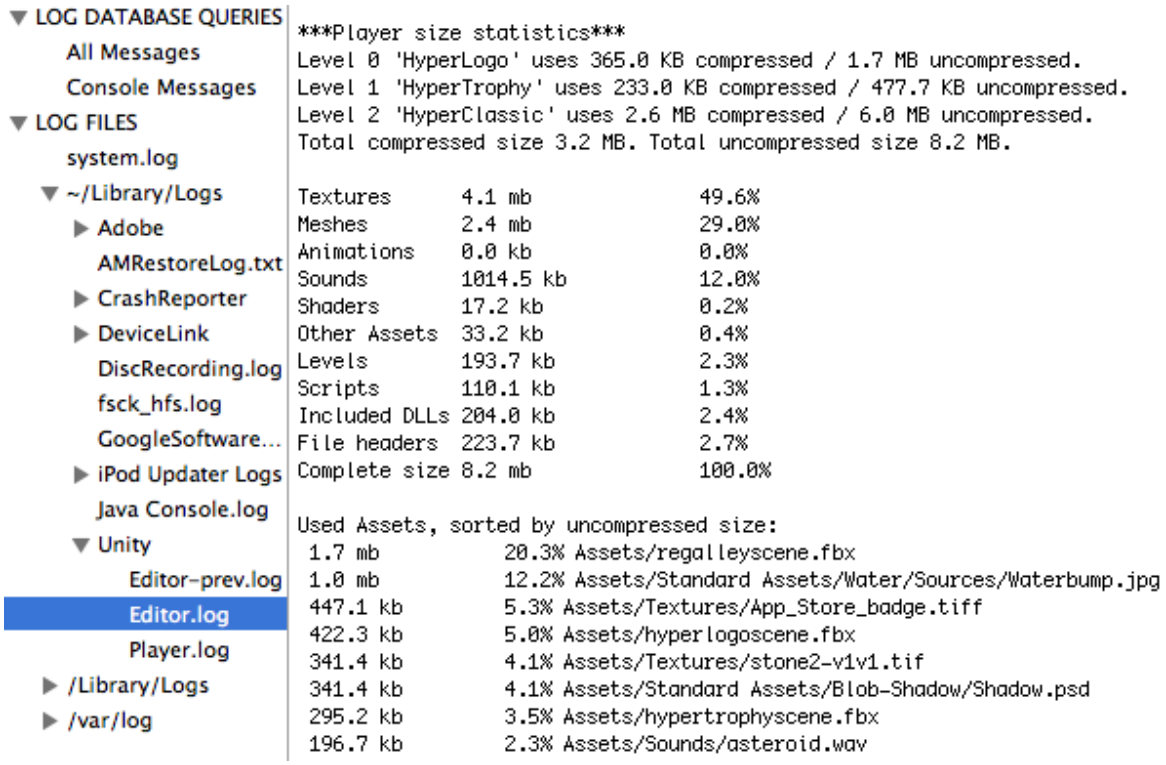
In addition, the Unity player is upgraded with every Unity release, and sometimes more often. The upgrades shouldn't break old content, but I've had web players break in some way every other Unity release. Plan on testing andd updating your web players with each Unity web player upgrade.

13.6. Size Optimization

Size optimization is particularly important in building a web player. There are plenty of places, like hotels and my local public library, that still take a long time to load a 5MB web

player. And think of it as good practice if you elect to develop for the iPhone or Wii (or any other console or mobile device).

After a build you can look in the console (Windows-Console or open the console externally from Application/Utilities) to see what went into your player.



In general you don't need to worry about assets that are not included in any scenes. Those assets won't be built into the web player. Scripts are an exception. They typically don't add much to a web player's size, but if you want to be thorough, remove any scripts that aren't used. .NET libraries can take up a lot of space, e.g. System.Diagnostics adds 700k. So don't link to any external libraries that you don't need.

Minimize textures by choosing the smallest acceptable size and the greatest acceptable degree of compression. Choose a format without alpha if you're not using it. Disable mipmapping if it's unnecesary, e.g. if you're using it as a GUITexture.

It's easy to forget that fonts can comprise a significant amount of texure uage. By default, fonts are imported as Unicode. Usually, ASCII is sufficient and sometimes just upper or lower-case.

Audio typically also takes up a lot of memory. Unity accepts OGG Vorbis compressed audio and will convert to that format, too.

Note:

When compressing audio, be sure to select Decompress On Load. It is recommended by the documentation for performance reasons (you generally don't want audio decompression to occur during gameplay) but the default is off.

13.7. Publishing

Any web page can host a Unity web player, so if you have a web site, you can set show your web player by copying the HTML and .unity3d files generated by the build process.

Shockwave is a well-known web game portal that mostly features Flash games but has been known to feature Unity web players. The submission process involves filling out an online submission form, uploading your game, and waiting to hear back.

Other web game sites that share ad revenue and support Unity games include, GameJolt, which is quite polished even though it's in beta, and Wooglie, which is not quite as polished and has a slower download speed, but is Unity-only, so there's less competition. Both GameJolt and Wooglie also feature convenient developer accounts for submitting and updating web players. EditorialD is another Unity-only site but doesn't feature developer accounts so you have to email them to submit a web player.

Kongregate has accepted some Unity games but apparently has backed off due to "confusion" about the plugin installation. Reportedly they will start accepting Unity players again in "early 2010".

You can set up your web player on your own as a Facebook app, but Dimerocker makes that easier by providing the necessary server and managing the ad placement, although you currently still have to perform some setup on the Facebook end.

Note:

Dimerocker is currently in beta and the revenue-sharing arrangement from ads is still TBD.

Unicade appears to be unmaintained since the advent of Unity 2.0.

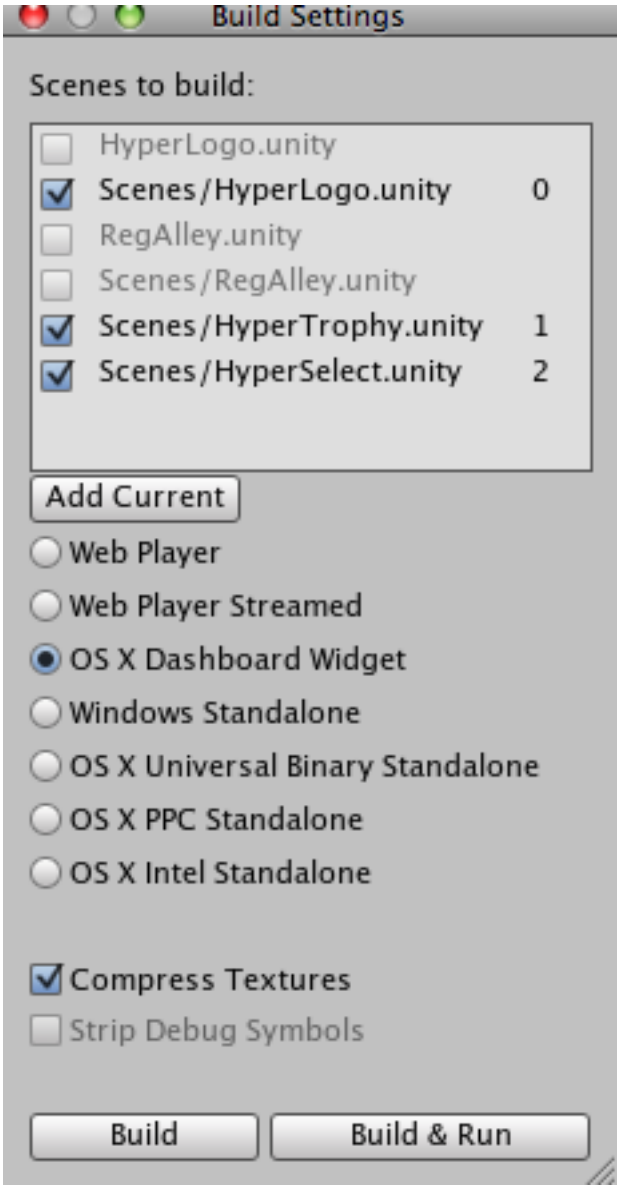
For comparison, you can view HyperBowl on these platforms:

- HyperBowl web site (self-hosted)
- HyperBowl Facebook app (via dimerocker)
- HyperBowl on GameJolt
- HyperBowl on Wooglie

14. Mac Widgets

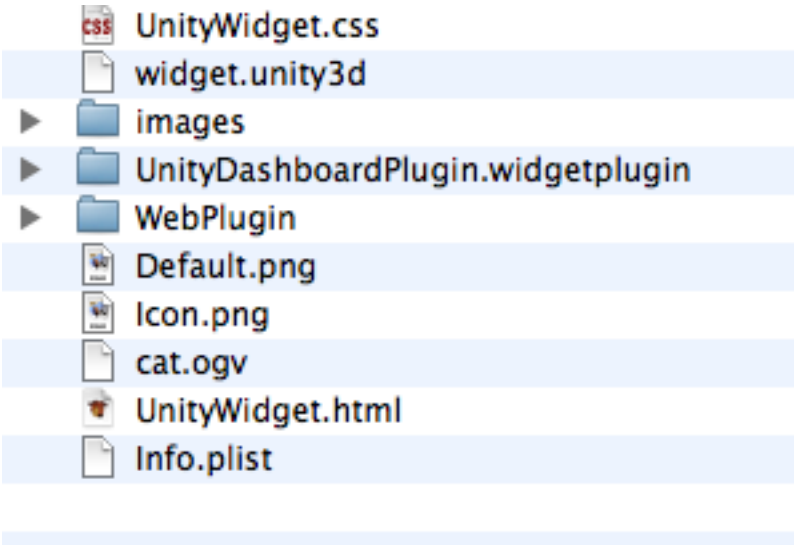
14.1. Building

Besides regular Mac apps, Unity can build Mac widgets, using the Unity web plugin.



14.2. Customization

The widgets are just a web player wrapped with the widget HTML. If you right/option-click on the .wdgt file and select Show Package, you'll see the generated Unity file and accompanying HTML, CSS and images. Any additional files loaded by the widget can be added in the widget. For example, this widget streams a movie, cat.ogv.



You will probably want to customize the default Unity build by replacing the Icon.png file, which is the icon that displays in the Dashboard dock. Version information can be adjusted in the Info.plist file. You can customize the back panel by modifying the HTML and CSS files.

14.3. Publishing

Most widgets are submitted and listed on the Apple widget download page. Most widgets are

free, though some are listed as shareware. If a widget is listed as a Staff Favorite or Featured Widget, it certainly ends up in the Top 50.

Widgets published by Technicat (Fugu Games) include HyperBowl Classic, FuguBowl, FuguMaze, FuguTilt, FuguFlip, Fugutype,

Other Unity widgets include 3D Paradise Paintball MacPinball, Banana Warehouse and Blingy.

14.4. References

See the Apple Dashboard developer article for an overview of widget and links to more articles.

15. Windows

15.1. System Requirements

XP, Vista

15.2. Packaging

I use NSIS with a reusable installer script to generate installers. See Windows versions of Fugu Games for examples.

15.3. Publishing

Steam is a possibility, by creating a plugin (see forum post). Example - The Graveyard. Also, Blurst.

16. Mac

16.1. System Requirements

MacOSX 10.3+, Intel and PowerPC

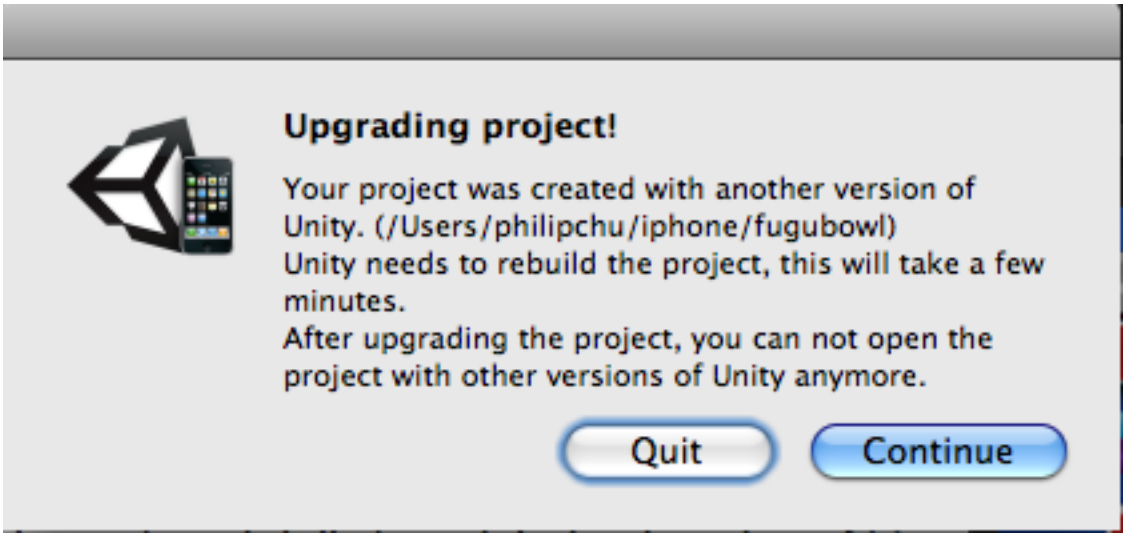
16.2. Publishing

Apple software site is a good way to advertise. Big Bang Games and Freeverse sell Unity games in retail. Check MacFun.

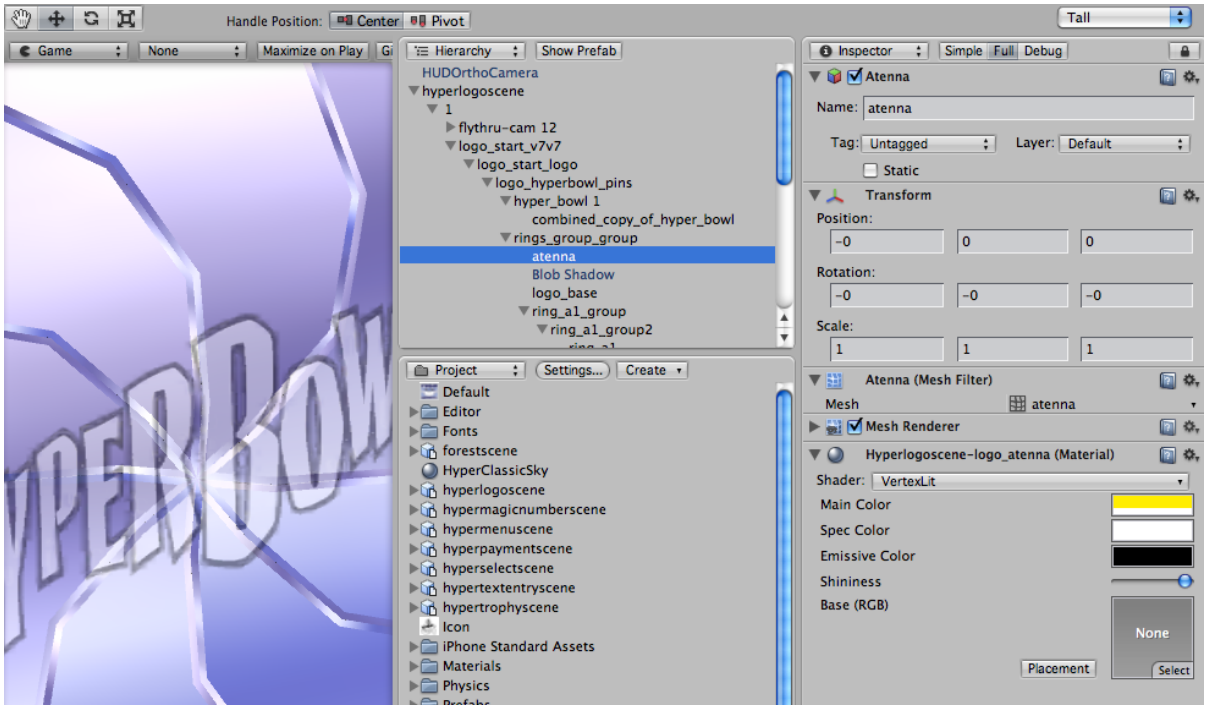
17. iPhone and iPod touch

17.1. Overview

Unity iPhone is a similar but separate product from Unity for Mac/PC, although it does require that you have a license for the Mac PC version. In fact, you can move a Unity project to Unity iPhone and vica versa, although you'll get an "upgrade" confirmation each time.



Unity iPhone does tend to lag behind the features of Unity (you'll notice Unity iPhone still sports the GUI of that Unity had before version 2.5), and there have been periods of incompatibility when new version of Unity were released.



Unity iPhone runs only on the Mac, since all iPhone apps have to be built by XCode, and the Unity iPhone publish step creates such an XCode project (and in fact Publish and Run actually attempts to initiate the build-and-run step in XCode).

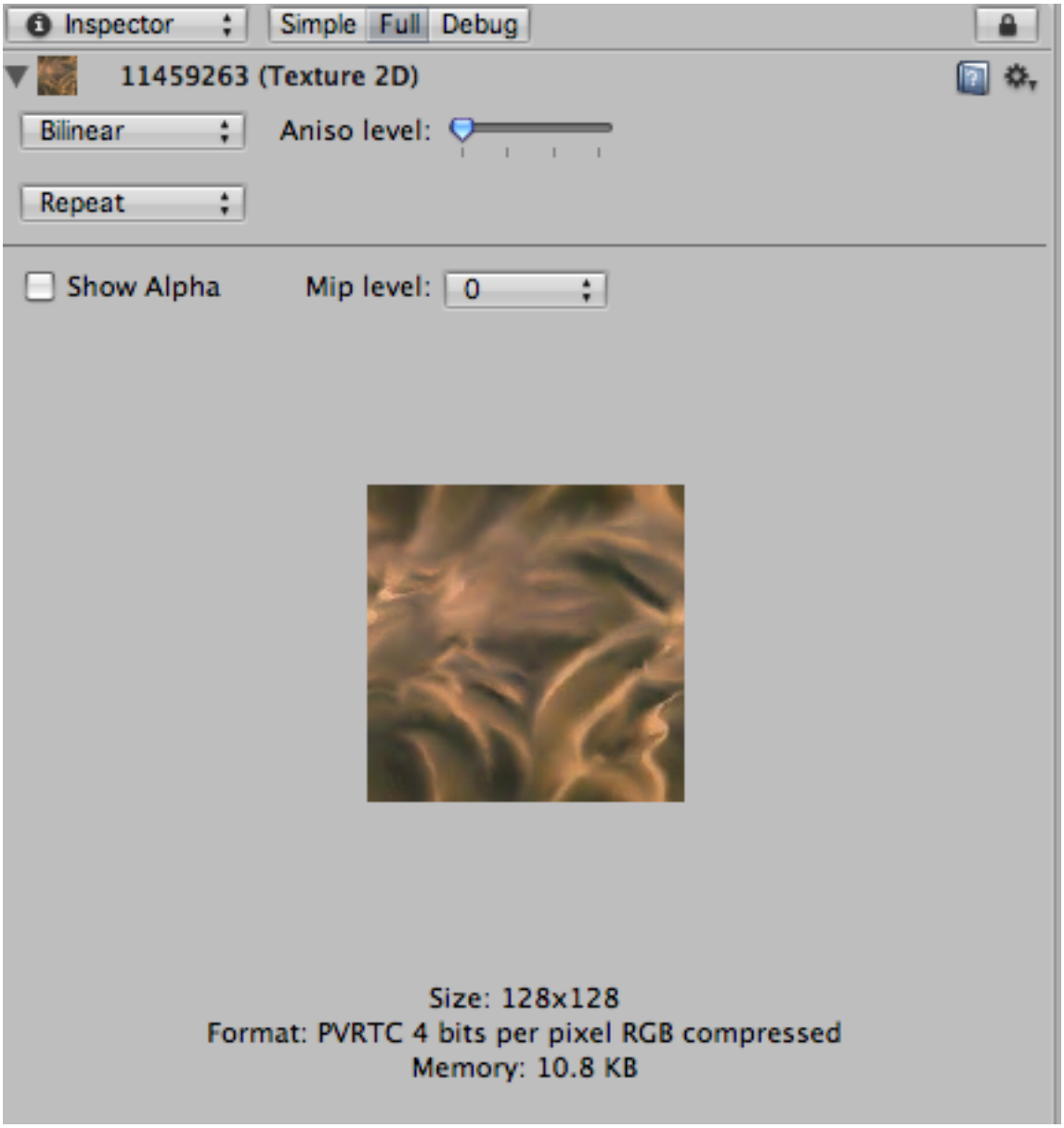
The nice thing about iPhone development is you don't have to worry about all those pesky PC video cards to test and support. The days of just one iPhone hardware spec are already over (see the Unity blog on iPhone 3G hardware) , but it's still a relatively fixed set, and you should be safe if you target the lowest common denominator

- screen resolution: 320x480 or 480x320, depending on orientation.
- RAM: 128MB
- texture memory: 24MB
- textures: RGB/A uncompressed and compressed in PVRTC format, max size 1024x1024
- screen refresh rate: 60Hz
- 3D support: OpenGL ES 1.1

17.2. Textures

Instead of DXTC compression, the iPhone uses PVRTC compression for textures. Unity will

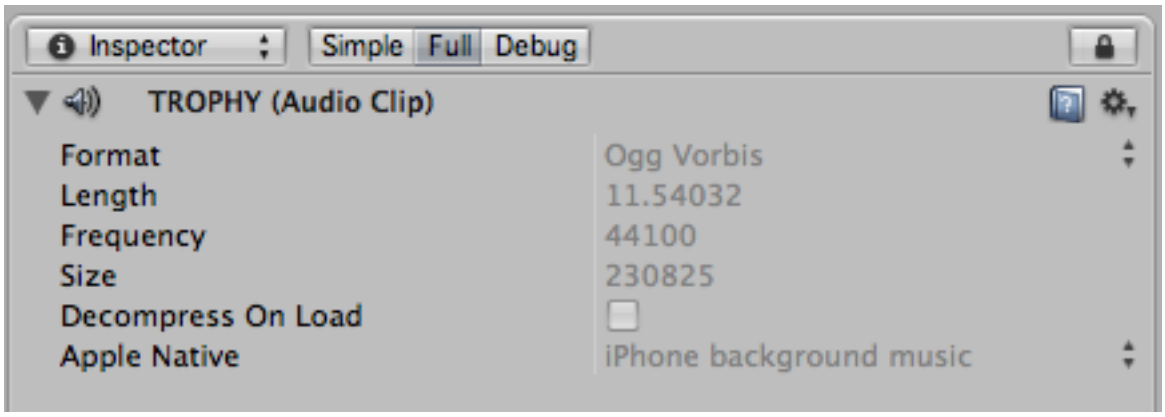
import textures to these formats, in increasing order of performance - 24-bit RGB, 16-bit RGB, 4-bit PVRTC, 2-bit PVRTC



17.3. Audio

Instead of OGG Vorbis compression for audio, the iPhone accepts MP3 (.mp3) and AAC

(.m4a) compressed audio.



An easy way to convert audio to AAC is to open it in iTunes, right-click select "Convert to AAC", right-click on the resulting audio clip to select "Show in Finder", then copy or drag the .m4a file into the iPhone project.

Note:

The iPhone hardware can play only one compressed audio at a time, which is why the setting is termed "background music".

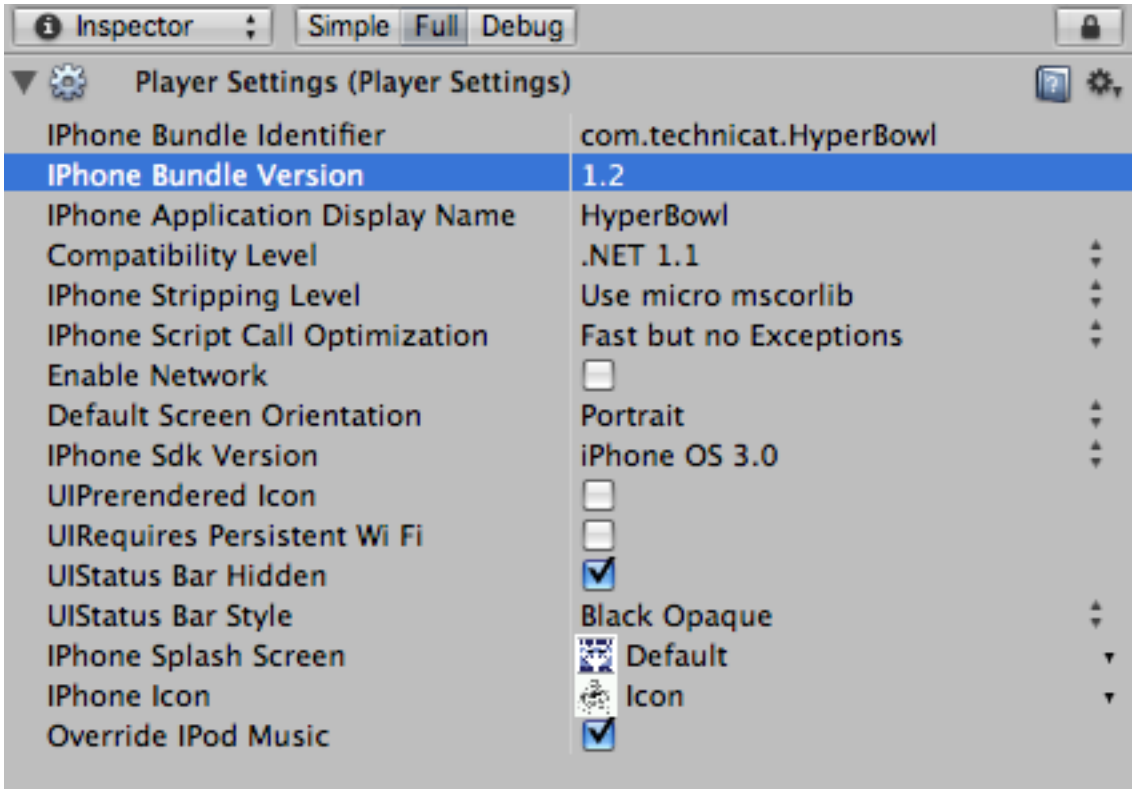
17.4. Scripts

(example of strict typing)

17.5. Optimizing Memory

This is where we separate the men from the boys, or PC developers from console developers (many a PC game developer was seduced by the PS/2 and eventually rescued by the XBox). Not only are you restricted by the available RAM on the iPhone, you're only going to get a piece of that and an unknowable piece of that. (see Noel Llopis' blog)

All of the space optimization tips for web players apply here (though of course you'll use the iPhone-appropriate texture and audio compression formats). Also, select the most aggressive code-stripping option feasible in the Player settings.



17.6. References

The iPhone section of the Unity manual is in the Components page, but not in the online manual. See also these online resources:

- Unity iPhone roadmap.
- iPhone technical specifications
- Developing Applications for the PowerVR MBX
- PowerVR MBX
- iPhone Developer Program
- Unity forum announcement
- Unity iPhone forum
- Unity iPhone apps
- Unity product page
- preflight application errors

- Apple Developer Forum
- iPhone Dev SDK forum (unofficiail)
- Getting Started with the Unity Remote

17.7. Features

Features specific to the iPhone version

- Occlusion Culling
- Orientation
- Touch input

17.8. Pro Features

- Build stripping
- Application.OpenUrl
- Replacement of Unity splash screen
- Asset streaming

17.9. Inherent Limitations

These limitations are unlikely to change

- OpenGL ES fixed function pipeline (no pixel shaders, expect performance impact with vertex shaders)
- no render to texture (no video, reflective water...)
- no terrain

17.10. Functional Limitations

Application.ScreenCapture and GL.ReadPixels are not implemented, which means no custom screen snapshot capability. Note that pressing Home+Power still works.

OnMouseDown, OnMouseOver, OnMouseUp are not implemented for performance reasons (raycasting is expensive), but you can roll your own, e.g. with this wiki script.

No video textures, although movies can be played separately.

No streaming data, including audio.

.NET 1.1, not 2.0 like Unity desktop (for size reasons)

17.11. Performance Tips

- Read the iPhone Optimization section of the Unity manual (which pretty much covers the rest of these tips
- General guideline: target 7000 static visible triangles (see Unity forum), 3000 animated
- Use PVRTC texture compression whenever possible, 2-bit whenever possible.
- Use occlusion culling
- Avoid the GUI system
- Disable the accelerometer if unused
- Use the most aggressive Player settings
- Minimize use of transparency
- Use the profiler

```
iPhone Unity internal profiler stats:
cpu-player>    min: 35.9   max: 111.6   avg: 45.0
cpu-ogles-drv> min: 12.9   max: 21.1   avg: 15.7
cpu-present>   min:  1.4   max:  2.9   avg:  1.7
frametime>     min: 65.8   max: 134.9   avg: 72.8
draw-call #>   min: 136   max: 203   avg: 151
tris #>        min: 7144   max: 8214   avg: 7393
verts #>       min: 14139   max: 19005   avg: 15274
player-detail> physx:  2.8 animation:  0.2 skinning:  0.0 render: 36.4 fixed-update-
count: 3 .. 7
mono-scripts>  update:  3.1   fixedUpdate:  0.0 coroutines:  1.3
mono-memory>   used heap: 524288 allocated heap: 643072
```

17.12. Build Errors

- IOException error during build - make sure you don't build within the Assets folder
- Make sure assets are imported into iPhone-supported formats - Unity Indie/Pro assets will usually need to be reimported.

17.13. Publishing

iPhone apps are only published on the App Store. Apple supplies the provisioning (i.e. you upload the app) and takes a thirty percent cut. App can be updated any time. Risk factor - they have final say over submission. Approval times range from a few days to a really long time.

During submission you supply the information that will appear on the App Store:

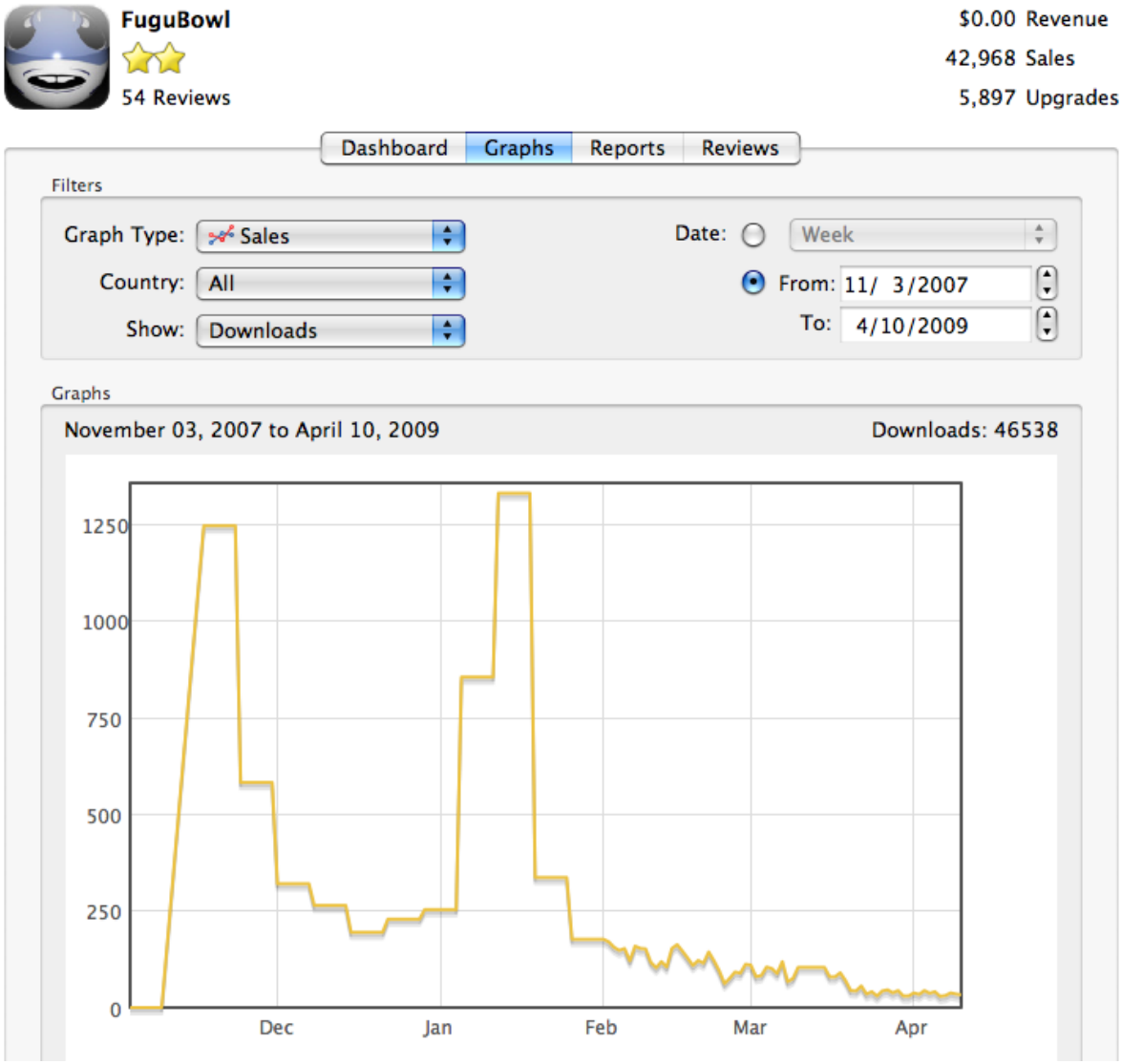
- a 512x512 large icon
- 1-5 screenshots (use XCode or the Home+Power button)
- release date
- price
- description
- version
- new release notes

After approval, you can edit this information.

17.14. Tracking

Once an app is approved, you can download app sales/upgrade statistics manually and/or use a tool.

AppViz costs $29.95 and downloads the latest stats and displays them with various graphs:



17.15. Promotion

The URL of the app can be obtained by right-clicking (control-clicking) various occurrences of the app icon or name in iTunes and choosing Copy URL. The same can be done for the "All applications by..." link. For example:

This badge is obtained from the Apple iPhone developer site and requires signing a use-policy agreement.

For each release of an app, up to fifty promo codes can be requested from Apple, each expiring in four weeks. Most review sites state they accept promo codes for their reviewers (and some even actively solicit them). You can also use promo codes to get some feedback and initial reviews. Note that posting them on a forum can get you both good and bad reviews, useful and annoying feedback.

Blogs:

- iPhone Marketing Resources
- iPhone App Ranking Tools
- 30 Review Sites to Promote Your iPhone Application

17.16. Tools and Libraries

The Unity iPhone Enhancement Pack adds miscellaneous functionality to Unity iPhone, including a second splash screen, photo picker, song picker, email composer, web view... See discussion thread.

SpriteManager has been used for many 2D iPhone games. A free versionis on the wiki, and a commercial version, Sprite Manager 2, is available.

There are at least three social gaming networks that provide Unity iPhone integration: Scoreloop, Agon, and OpenFeint. They provide features such as leaderboards, challenges and achivements.

18. Wii

18.1. References

- Unity Wii Publishing

18.2. System

Architecture is similar to GameCube, apparently

18.3. Publishing

WiiWare. Self-publishing options is not clear, but it's certain that Nintendo has final say over

submission.

Unity charges per-title, $30,000 for a retail game, $15,000 for a WiiWare title, and requires becoming an authorized Nintendo developer, which apparently is problematic without an industry track record and secure (as in zoned for business) office.