Adopting Game Engine Technology for Rapid Virtual Store Visualization

Darren Brooker – Autodesk

AV3413

The process of designing for retail can be both expensive and time-consuming. This class shows how building stores in a virtual environment reduces the risks associated with physical store mockups, helping to drive both efficient problem-solving and the rapid design iteration required that together can help your customer meet the requirements of the rapidly-changing retail landscape. By taking full advantage of a workflow for moving large-scale architectural projects from Autodesk® Revit® software into a game engine environment using a streamlined approach, this class aims to demonstrate how Unity not only acts as an efficient design tool for store and space layout planning, but has the potential to act as a sophisticated and valuable Demands Insight Model.

Learning Objectives
At the end of this class, you will be able to:

- Explain the benefits of a streamlined workflow for working with large-scale Revit schemes in Unity
- Import large datasets from Revit into Unity and explore the environment in real time
- Grasp the basics of authoring in the Unity environment and understand its possibilities
- Export a Unity environment to a mobile device or to a portable executable file

About the Speaker
Darren Brooker has well over a decade's experience working in CG and before jumping ship to Autodesk in 2004, he worked in various roles, mainly as a texture artist and lighting Technical Director. Darren's first job at Cosgrove Hall involved adding CG elements to stop-motion backplates for Andy Pandy, and after this he moved on to various UK architectural, games and post-production facilities. During 2004 he was part of the team at Red Vision that won a BAFTA for Best Visual Effects for their work on the BBC's Battlefield Britain. Today, Darren manages a technical team of seven technical specialists who between them cover the entire Autodesk M&E portfolio and interface with our key customers, helping them ensure their pipelines are ready for tomorrow's challenges.

darren.brooker@autodesk.com
@darrenADSK
http://area.autodesk.com/blogs/darren
Introduction: From BIM to DIM

BIM is widely understood – both in terms of its meaning and the value it brings – by those involved in the construction & management of the building right the way from concept to completion and beyond. The purpose of today’s class is to show how the value of BIM can be leveraged further downstream to drive a Demands Insight Model – or DIM.

So what the phrase “from BIM to DIM” really means is how the Building Information Model can be leveraged to bring additional benefit to the manufacturers of the products stocked in large retail environments in terms of data intelligence.

Retail is an increasingly complex landscape for both manufacturer and consumer alike. The increasing trend for consumers to make their purchasing decisions at the shelf means that product manufacturers are looking to understand how they can improve their product configurations and presentation in order to drive increased sell-through.

Retailers and product manufacturers already rely on detailed data intelligence around consumer behavior to build compelling offers and in-store marketing of their products. In this session we’ll go one step further and look at how by using game engine technology, the intelligence that can be gathered has the potential to give valuable insights that can model consumer behavior all the way through to the point of purchase. We’ll look at an outline proof-of-concept workflow for gathering meaningful upfront data intelligence and look at its potential for agencies working in the retail & product manufacturing spaces.

Step#1: Revit

In this class, Revit is just the starting point and it’s from here that we will be looking at how we take the Building Information Model for use as the basis for our virtual store environment. We’ll use game engine technology in the form of Unity to show how retailers can reduce the risks associated with physical store mockups, helping to drive both efficient problem-solving and the rapid design iteration required that together can help product manufacturers meet the requirements of the rapidly-changing retail landscape.

Obviously we’ll avoid making changes to the Revit model specific to our workflow into the Unity game engine. Instead, we’ll use 3ds Max Design as the solution that acts as a hub for assembling the content that will make up the game environment.

Step#2: Autodesk 360

The proof-of-concept workflow that you’ll see today for building a Demands Insight Model is intended to be a small-scale model that enables an iterative workflow without the large-scale investment in expensive fixed assets like render farms and immersive display technology. We’ll touch on rendering out of 3ds Max Design, but at this point it’s worth noting that Autodesk360 offers an elegant cloud-based solution to rendering that gives everyone access to infinite computing power, as well as the ability to share files which brings benefits when working with extended teams that would be typical of retailer/manufacturer project team. Additionally,
Adopting Game Engine Technology for Rapid Virtual Store Visualization

Autodesk360 includes access to 123D Catch, which offers a simple and powerful solution for reality capture that we'll explore a little later. To render in the cloud, it's as simple as you might imagine: one simply selects *Render in Cloud* from the *View* menu and selects the 3D View to render and render quality / output size. Options for email notification of the completed render and to continue the render in the background mean that you can carry on with your work…

![Render in Cloud](image)

…until the email notification arrives, when one needs simply to refer to the *Render Gallery* for the latest rendering, which is viewed in context of all previous renderings on Autodesk 360.

![Render Gallery](image)
Step#3: 3ds Max Design
We'll be utilizing the Revit model within 3ds Max Design as the basis for preparing the Revit model. It can be thought of as the hub, where all additional content will be assembled ready for export directly into the Unity game engine. The first step is to use Revit’s one-click workflow for export to 3ds Max Design, which can be found under Revit > Suite Workflows > 3ds Max Design Interior Rendering. Before we export the scene, it’s important to understand that if we want a camera view to come over from Revit, then this view needs to be open in Revit. In this case, in Revit we’ll open the 3D View that we rendered in the cloud – the one entitled 3dsMaxView – before we export, as this is the view that’s important to our workflow, as its shows the end aisle – gondola ends or encaps to use the term used in retail – where the promotional display will be hosted.

All the settings can be kept at their defaults as the model is exported directly to 3ds Max Design. When 3ds Max Design opens, you will see that the model is imported with the daylight system, all lights and a single camera. If you have the bottom-right viewport active and hit the C button, you will see the 3dsMaxView camera view is active. The reason this looks slightly different is that the resolution that the 3dsMaxView camera is set to render at in Revit is 1,000 x 749 pixels, so you need to take two steps to rectify this:

1. Hit F10 to open the Render Setup dialog and enter this same resolution
2. Click the 3D View label in the top-left of the camera viewport and enable Show Safe Frames

Now your 3ds Max Design viewport should match that of Revit:
We have a few more changes that we need to make to the scene in terms of its lighting and associated render settings, as well as the scene’s materials before we can consider rendering the scene, or starting to transfer content to the Unity environment. We’ll look at the lighting first, starting with the natural lighting; so select the SunAndSky-002 object in your scene (hitting H will bring up the Select by Name dialog where such selections can be streamlined). In the Modify panel, check the Active box next to the Sunlight component to turn on the Sun. Just below this area of the Daylight Parameters rollout, change the position radio button to Date, Time and Location and hit the button marked Setup…

For this example, we’ll set the date and time arbitrarily to 10:00am on 21 February 2013, because this is when the light is low enough to penetrate the building at the far end as seen from the camera view. Set the location to London: another arbitrary choice which would be determined by the physical location of the building and would more likely involve the use of longitude & latitude values. As we have set the time of day to be 10:00am, it makes sense that we turn off the lights to the exterior of the building. This could be done using the Select by Name dialog again, but the best tool to do this in is the Light Lister, which can be found under the Tools menu. From here you just need to uncheck one check box next to the lights whose names begin StreetLight and External and uncheck the On checkbox.
Adopting Game Engine Technology for Rapid Virtual Store Visualization

You are almost at the point where you could render a still now, but if you did you’d find it dark as your **Environment** controls are set to **Outdoor Daylight** – which is somewhat peculiar given that the one-click workflow from Revit was the Interior Rendering. Anyway…correcting this is simple, just press the 8 button, or choose **Environment** from the **Rendering** menu. Change the **Exposure** value to **Physically Based Lighting, Indoor Daylight** and you should see your viewport’s lighting reflect this change. Before testing these values in a lengthy render, the best place to validate your settings is in the **Exposure Control** rollout of the **Environment & Effects** floating dialog box, so hitting the **Render Preview** button would give you the result you see below.

Now, whilst the overall exposure might look at roughly the right level, there is a strange green tint to the exterior as seen from the camera and this brings us from render settings to materials. If you hit **M** – or choose **Rendering > Material Editor** to bring up the **Material Editor**, you can start to look at the scene’s materials and work out where you need to make some changes.
The *Scene Materials* section on the left of the *Slate Materials Editor* lists all the materials used in your current scene, so from this list find the one referenced in the *Environment Map* slot of the *Environment & Effects* dialog box: *Map#90* (*mr Physical Sky*). Drag this from the list on the left into the gridded viewing area to the right and you can spot the first problem with your Revit materials. As you can see below, *Map#89*, which is connected to the *Haze* parameter of this material, is showing up red in the preview: a sure-fire sign that something is wrong, as this should display a thumbnail of a hemispherical sky image.

Right-click the *Haze* map slot and choose *Clear* to get rid of *Map#89* in this material. Now click the blank slot and choose *Bitmap* from the *Maps > Standard* rollout of the resultant *Material/Map Browser* dialog. Navigate to the same place that the hemispherical sky image was stored — the location on a standard install is: `C:\Program Files (x86)\Common Files\Autodesk Shared\Materials\Textures\Environments\SunAndSky\Presets\5`

Load the *hemispherical_003_half_tiled.exr* image and check that this is allocated the correct mapping coordinates: the *Mapping* drop-down should read *Spherical Environment*. As this image represents a sky dome, it should only appear in the upper half of the slot, so you need to set your *U Tiling* to 2, and to keep the image in proportion you’d need to set the *V Tiling* value to 4. Check the *Tile* checkbox for the U value and enter 0.1 in the *V Offset* field. Now the sun in the hemispherical sky image lines up roughly with where the sunlight assembly is in the scene and the sky’s correctly proportioned. When the settings match those below, you should then return to the top level of the material and drag *Map#91* from the *Haze* slot to the *Use Custom Background Map* slot so that this renders in the background.
Adopting Game Engine Technology for Rapid Virtual Store Visualization

The final thing you need to do is uncheck the Process Background and Environment Maps within the Exposure Controls rollout of the Environment & Effects dialog box so that this renders uncorrected. (_AUsupermarket004.max)

You’re now at a stage where a test render might be appropriate and if you rendered now, with the default render settings, you’d get something that resembles the following image:

What this rendering highlights is problems with the materials that have come over from Revit. The green haze on the ceiling shows a problem with the glass material and if you look closely
you can see problems with the steel material on the roof members too. You could manually correct all these material problems, but one of the benefits of 3ds Max is it’s community and the amount of third-party plug-ins developed for all kinds of tasks from rendering and visual effects through to simple things like converting materials. A €59 purchase will save a lot of time here: for V-Ray users the plug-in from www.3dstudio.nl takes one click to do the conversion, and for conversion to Standard materials – which we’ll need for the Unity environment – this takes two clicks. (_AUsupermarket005.max)_

This corrects the green effect that the Revit glass was bringing to the scene and apart from the floor works well. Given that we don’t need the look to match exactly, the two-clicks have got us a long way towards our goal of a scene we can export and if you render again, you can see that this renders 5x faster than with the previous materials.
The problem we have with the floor is one of incorrect UVW mapping coordinates on the floor: we need to take one step to understand why the map on the floor is being displayed incorrectly: first choose **Customize > Units Setup...** and select **Metric** as the **Display Unit** scale. In the **Material Editor**, if you navigate to the **MultiMat_19** material that is applied to your floor and click through to the lower of the two materials – the one with the **ID** of 2 – and click through to the **Bump** map, you can see that the map is set to be 4.0m x 4.0m which looks correct given the image. However, given that this is displaying incorrectly in the rendering, this points to a UVW mapping coordinate problem.
To take this a stage further, you'd need to start thinking about increasing your Final Gathering settings to something like those below…
...however, this is not really the focus of our session, so we need to start to focus on getting assets assembled in this scene and out into Unity. (_AUsupermarket006.max)

**Step#4: 123D Catch**

We’re going to take a sidestep into 123D Catch for one moment to explore the possibility that reality capture could offer to a manufacturer with a product still in development. We’ll start with the free reality capture app 123DCatch, which works well in the cloud via iPad & iPhone cameras. By taking a few photos of your prototype using your mobile device, you can automatically upload these for processing in the cloud. For example, taking multiple photos –
Adopting Game Engine Technology for Rapid Virtual Store Visualization

between 20 and 30 ideally – of this physical bottle prototype would take a few minutes to return a 3D mesh.

Once the resultant mesh is processed, it appears on the iPad for validation via the usual methods of simple touchscreen navigation. This mesh can then be emailed as a .3dp file directly from the iPad. This file can then be opened on the desktop using 123D Catch PC, where you can ensure the origin and coordinates are set correctly and that the scale is correct before you export it as an .fbx file to bring into 3ds Max Design.
When imported, the geometry requires clean-up, as the full mesh is imported, including any background elements like the table, as well as all the cameras that represent the position of your iPhone camera at capture.

Once cleaned up, this can be saved out as a 3ds Max Design file ready to be brought into the master scene file in 3ds Max Design, ready for subsequent export to Unity. 123D Catch is free to use and gives you unlimited cloud storage to save your creations in the cloud. Recap, included with the Building Design Suite gives you more professional tools for dealing with both image-based reality capture and point cloud data. Things like the ability to manually stitch images, add survey points or known distances for improved precision means that this is a far better fit than 123D Catch for professional application.

**Step#5: 3ds Max Design**

We’re now at the stage where 3ds Max Design’s role becomes the hub for assembly of the components that will make up the game environment in Unity.

The first thing we need to do to set up our promotional end aisle is to replace the merchandiser we can see through the camera view to one custom made to fit our product. It’s not possible to delete the one in the scene as it’s brought into 3ds Max Design as *Linked Geometry*, so the easiest thing is to select it and right-click, choosing *Object Properties*…

Change both the *Rendering Control & Display Properties* options from *By Layer* to *By Object* and uncheck all the boxes, as shown below:
Next choose *Hide Selected* from the Display panel. If you then choose *Import > Merge* from the 3ds Max Design button menu in the top-left, select the `_AUMerchandiser.max` file and bring in the one object from this file, choosing to *Use Scene Material* when prompted about the duplicate material. This gives us the merchandiser that is configured to fit our new product. Repeat the import process, this time selecting the `_AUBrainballsBox.max` file and selecting the *brainballs001* object to merge. With the *brainballs001* object selected, right-click the *Select and Move* button in the top toolbar and position the cereal at $X: 5.075m / Y: 26.05m / Z: 1.45m$, which should place this in the top-right of the new merchandiser. Now choose the *Array* tool from the *Tools* menu and position 42 instances of the *brainballs001* object across the merchandiser. If you hit *Preview*, you can see that the values below work perfectly…however, hit cancel at this point.
The reason we do this is because we’re going to use a referencing system to bring in the goods across the store, so instead with the brainballs001 object selected choose Save As > Save Selected and save this as _AUbrainballsBoxPosition.max. Now delete the brainballs001 object from your scene and choose References > XRef Objects from the 3ds Max Design button menu. Click the Create XRef Record from File button in the top-left and select the _AUbrainballsBoxPosition.max file you just saved. Now close the XRef Objects dialog and repeat the array options you entered above, this time clicking OK to complete the operation.

For our purpose this workflow is adequate, however, this is where you’d have to invest some time and effort building a robust link between your store inventory and the virtual store if you were doing this on the scale that retailers operate. 3ds Max is extensible via MAXScript, so there’s no reason why you couldn’t hook a database of products to a database of models that are imported with customisable configurations. We just won’t be doing that today.

One final thing we’ll do before we jump over to Unity is to show how flexible it is to work with externally referenced assets using XRef Object. We’ll repeat the process above, this time bringing in a single bottle of our new product via the XRef Object feature: first we select the _AUzombieWaterPosition.max file, then we use the Array tool as below:

Once this has been done, by right-clicking the _AUzombieWaterPosition.max entry in the XRef Objects dialog and choosing Merge in Scene, it’s simple to bring these objects directly into the master scene. (_AUsupermarket007.max)

By then using the Select by Name dialog, it’s simple to select all 124 zombieWater objects and by using the Save As > Save Selected feature, these can be saved to a separate .max file before being deleted from the master scene file. We’ve also done this with the brainballs array of objects ready to export to Unity via .fbx. The most streamlined method for ensuring that the elements we need to bring into Unity get brought in with all the materials in tact is to export them twice: first with the materials embedded in the .fbx file, then with no material embedding. From our _AUsupermarket008.max file we choose Export > Export from the 3ds Max Design button menu and name the file AUsupermarket008Embed.fbx. The two options that are important at this point are the Embed Media checkbox and the Units, which need to be set to convert to
centimetres to scale up the scene’s objects by 2.54 so that the scales are correct in Unity, which is important as for the physics engine in Unity to work correctly the scene has to be to the correct scale.

Once the export is complete, repeat the export, but this time uncheck the Embed Media checkbox and name the file AUsupermarket008.max. (AUsupermarket008.max)

Of course, it’s simple to generate photoreal planograms from the orthographic viewports in 3ds Max Design once your shelves are populated, so retail buying teams and operational management can also reap the benefit of a virtual store approach in terms of time and cost.
Step#6: Unity

At this point we’re going to take a leap forwards and start to build up our Unity scene from the various components assembled in the master 3ds Max Design scene file. Upon starting Unity, you should be faced with a relatively familiar environment. In short there are five areas of the Unity UI that you need to be aware of to get started:

1. **Scene View** (top-left viewport): Where the game is visually constructed, by manipulating objects in 2D & 3D. Manipulation of objects in this window is carried out via the four buttons above the Scene View viewport on the left-hand side. These carry the familiar QWER keyboard shortcuts to the viewport/position/rotate/scale controls. Beyond this, axis helpers and a device similar to the View Cube should ensure that this is not too difficult to ramp up on.

2. **Game View** (bottom-left viewport): Where you preview your game at any time. The big Play button in the middle of the screen right below the menu bar takes you into a live preview of your game at any point.

3. **Project**: Shows all assets you’re that have been imported into Unity, so can depending on how you choose to manage your files, could reflect multiple projects, or in this case just the single AU project.

4. **Hierarchy**: Shows the subset of assets at the Project level that have been brought into the Scene View and are listed here in alphabetical order, with parenting represented as part of the hierarchy.

5. **Inspector**: Shows the properties of whatever you have selected across the Unity environment.

To import your first asset, choose Assets > Import New Assets… and select the AUsupermarket008Embed.fbx just exported. This will take some time, as it’s bringing in all the embedded material files, as you can see in your Project tab when complete. At this point, right-click the AUsupermarket008Embed object in the Project tab and choose Delete. This will leave two associated folders left in your Project tab. Now repeat the process, this time choosing to import the AUsupermarket008.fbx file, which is about half the file size. This file will now automatically link itself to all the associated media from your last import, saving you having to do this.

When this is complete, you can drag this from the Project tab directly into the Scene View, but to keep the origin at 0,0,0 you should drag it into the Hierarchy tab. After doing this, it should appear in both your Scene View & Game View, with the materials linked correctly:
If you were to press the Play button now to take you into the game preview, nothing would happen, as there is no game controller in the scene yet, so let’s set that up. In the Project tab, from the All Prefabs selection, choose the First Person Controller and drag this into the Hierarchy view, at which point your view in the Game View changes reflecting the view from your new controller. With the controller selected, go to the Inspector tab and change the Local Position values to X:-8.0 / Y:1.2 / Z:-30.0 and your Game View should roughly correspond with the previous camera views in Revit & 3ds Max Design. Hit the Play button now and you will see for the first time the effects of Unity’s physics engine as your controller plummets through the floor. Press the Play button again to stop the game preview and return to the editing environment. The problem is that no collision objects have been set up, so let’s do that. Back in your Project tab, select the AUsupermarket008Embed object and move to the Inspector tab. You should see a checkbox marked Generate Colliders, which simply needs checking. Hit Apply and a little while later when Unity has completed this operation, you can click Play to experience your Revit model as a game for the first time.

Using the W & S keys to move forwards and backwards and the space bar to jump, you can run round the aisles and jump on the merchandisers just like you were in a game of Unreal Tournament! The first problem that you see is that despite your environment being set up to the correct scale, your character’s movements suggest otherwise. This is purely because the First Person Controller is set up with quite heroic proportions. With this object selected, you can see in the Inspector tab that the height is set to 2m, and if you look in the Scene View, you can see that the camera is set quite close to the top of this model, so rather than reduce the height of the controller, which would interfere with the game physics, in the Hierarchy tab, expand the
hierarchy for the First Person Controller and select the Main Camera object. Now just move this down in the Scene View window until the view through your Game View looks more human!

Similarly, the controller’s motion is too fast, so still in the Inspector tab expand the Character Motor (Script) rollout and in the Movement section, half the values for the following fields: Max Forward Speed / Max Sideways Speed / Max Backwards Speed / Max Ground Acceleration. In the Jumping section, half the Base Height / Extra Height fields. Now your navigation will feel much more natural.

Back to the scene’s objects and you can see that you actually have two merchandiser units in place in front of your character, as the one you hid in 3ds Max Design has come over to Unity too, so select & delete the original one: the full name of this object is Merchandiser - Hussford - Medium 2299 [410358]

Similarly, as you’re not working with Linked Geometry any more, as you were in 3ds Max Design, you can delete unnecessary geometry, which will ease the load on the game engine. Unity’s a robust engine capable of dealing with big datasets, but every little helps. As you’d expect from a Revit model, there is some geometry that isn’t really needed in the game…the foundations for instance. To delete these elements, in the Inspector tab, type pile into the search box at the top of the list, right-click and select Delete. You can ignore the warning about breaking the connection to the prefab: this is just Unity being overly polite. Once you’ve done this you can repeat it for the objects with names that start with Concrete or Slab Edge.
Now let’s start to populate the store. Import the `AUbrainballsEmbed.fbx` file and after the full import of its associated files has been completed, delete the `AUbrainballsEmbed` object and import the `AUbrainballs.fbx` file, check Generate Colliders and hit Apply. If you drag this into the Hierarchy tab, this should position itself perfectly on the merchandiser we set up specifically for this product. Repeat this process with the `AUzombieWater.fbx` file (remembering to import the `AUzombieWaterEmbed.fbx` file first) and your merchandiser should now have two products on display. Within the Hierarchy tab you can drag both the `AUbrainballs` and `AUzombieWater` objects onto the top-level `AUsupermarket008` object, as they should live within the supermarket hierarchy.

Once this process has been repeated with the `AUcerealsAisle.fbx` file, and the merchandisers that these aisles replace have been deleted, you should have an idea of how straightforward this process is within Unity. With the `AUcrispsAisle.fbx`, `AUsoupsAisle.fbx` and `AUvitaminsDisplay.fbx` files imported, the end of your store where your promotional end aisle display is located is populated.

Now, it’s all been pretty easy so far, and that’s great, but you can only get so far in Unity without scripting. Let’s take things a little further and give our First Person Controller the ability to further populate our store: think of this as a gun that fires groceries. You’ll first need to define the bullet for this gun, so select one of the original brainballs boxes – cereals001 for example – and drag it out of the `AUsupermarket008` group so it exists at the top level of the hierarchy. Rename this as `brainballsBullet` so it has a unique name that tells us explicitly what it does. With this selected, go to the Inspector tab and at the bottom, hit the Add Component button and choose Physics > Box Collider.
Now repeat this operation and add a *Physics > Rigid Body* component. These two components combine to interact with the physics engine of Unity: the first gives it a simple box-shaped collision object, the second ties it into Unity's gravity.

If you hit play now, you will see your new *brainballsBullet* object drop to the floor, so you may need to position it somewhere in the scene where you can see it drop clearly.

If this were a game and the box were a bullet, you would generally want the bullet to self-destruct after a short while to avoid it interfering in your game environment, so for this you need
to go a step further beyond standard components and into JavaScript. Adding the script is simple enough: again use the Add Component button and this time select New Script, call your new script destroyBullet and click Create and Add. When you’ve done this, your brainballsBullet object has a new script called Destroy Bullet. Locate this script in your Project tab, and with it selected go to the Inspector tab, where you can see a preview of the code. Click the Open… button in the top-right to open this in the Mono editor.

As you can see below, the new script basically has a definition as to when the script starts and ends, but little else. Let’s take the destroyBullet script as a simple example of getting started with Javascript in Mono. If you look below, the first difference between the two scripts is the following line:

```javascript
var blt: float = 3;
```

This simply defines a variable named blt (bullet life time) and assigns it a float value, with a default setting of 3. This does nothing until it’s called within the function Start section of the script. The next difference is on line 5, where the following line appears:

```javascript
Destroy (gameObject, blt);
```

This calls the standard command Destroy and applies this to the gameObject after the value defined in the blt variable. The gameObject value is a simple reference to the base class of all objects in a Unity scene, so means destroy the base object (in this case the object that the script is attached to).
If you make this change to your script and save it in Mono, when you go back to Unity, your `brainballsBullet` object will have an entry tagged `Blt` with a default value of 3, that you can use your mouse to slide the value up and down. Hit `Play` now and the `brainballsBullet` object will drop to the floor under the influence of gravity, collide with the environment and destroy itself after 3 seconds.

Now, to use these as bullets, ordinarily you’d need a gun, so let’s set up a simple primitive as a gun. From the top menu bar, choose `GameObject > Create Other > Cylinder`. Select it, go to the Inspector tab and rename it `Gun`, changing the `Local Scale` to X:0.25 / Y:0.5 / Z:0.25. Now move it relative to your character so that it appears at the bottom-right of your `Game View`, so that it appears like you have a big gun designed for firing boxes of cereal.

Now drag it onto the `Main Camera` (which in turn is a child of the `First Person Controller` object) to make it a child of the camera. Next, create an empty game object (basically a blank asset) by choosing `Game Object > Create Empty` from the main toolbar. With this selected, in the `Inspector` tab, it is possible to give this a tag to make it visible in the editing environment. Click the small colored cube in the top left of the `Inspector` tab to choose an icon. Now rename it `gunSpawnPoint` and move it so that it’s oriented with the cylindrical `Gun` object, with the blue axis pointing out of the end of the gun.

Now to tie this all together with another JavaScript. With the `gunSpawnPoint` object selected, click the `Add Component` button and choose `New Script`, call your new script `shootBullet` and click `Create and Add`. Double-click the `shootBullet` script icon in the new rollout (in the `Inspector` tab still) to open this blank script in Mono. At this point we’re going to add a pre-defined script to the blank script, but you can read through the script and understand what it does:
Back in Unity, select your `gunSpawnPoint` object and in the *Inspector* tab you can see a new rollout. What you need to do is assign objects to the fields containing the *None* value. For the `BulletPrefab` field, you need to drag an object from your *Project* tab, but you have one small problem: your `brainballsBullet` object does not appear there. You could select it from the scene using the small circle icon directly to the right of the field, however as you set a script to destroy this object after 3 seconds, if you use it you’ll only be able to fire bullets for the first 3 seconds, at which point the cereal box acting as your bullet will have been destroyed. However, if you define the bullet object as a prefab, as we will do in one second, you are basically instancing it from this prefab each time you fire the gun.

Let’s do just that. By dragging the `brainballsBullet` object out of the *Hierarchy* tab into the *Project* tab, you are defining a *Prefab*, which means a component that can be referenced (and hence reused) many times. Once you’ve dragged it to the *Project* tab, you see at the bottom of this tab that it is named `brainballsBullet.prefab`. With the `gunSpawnPoint` object selected once again, drag the `brainballsBullet.prefab` object to the `BulletPrefab` field. For the `Shootclip` field, you can drag any .wav file, so in this case we’ll choose a Howa Model 1500 rifle. The `Mygun` field doesn’t require reuse, so you can use the circle icon to the field’s right to select from the scene directly the `Gun` object.

Finally, for the audio to function, you’ll also need to assign an *Audio Source* component to your `gunSpawnPoint` object. Do this via the *Add Component* button and choosing *Audio > Audio Source*. When you hit the *Play* button now you should be greeted by the joyous feeling of having created your own game (it’s the audio that really seals it!)
Conclusion
Now you’ve had a whistlestop tour of Unity and how easy it is to get assets into, you should have an idea of how straightforward it is to start to prototype gamified experiences such as the real-time walkthrough capabilities of this example. In this class we looked at a proof-of-concept as to how you might start to leverage 3ds Max Design & Unity together to deliver a model for gathering data intelligence in a visually-driven virtual store environment. A serious implementation into a large retail environment would require some investment in setting up an intelligent link between the inventory database and the physical products in Unity, but is a workflow that would be entirely possible. Along similar lines, if database links were developed, then it’s not a huge stretch to also imagine the overlay of real-time sales data onto the product at shelf level: with rate of sale, margin contribution and so on displayed across the store’s inventory, all displayed visually in an easy to interpret in-context environment.

Please don’t forget to check out my blog:
http://area.autodesk.com/blogs/darren
and follow me on Twitter: @darrenADSK

Thanks for attending!